A BIOLOGICALLY PLAUSIBLE SPARSE APPROXIMATION SOLVER ON NEUROMORPHIC HARDWARE

A Dissertation
Presented to
The Academic Faculty

By

Kaitlin Lindsay Fair

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2017

A BIOLOGICALLY PLAUSIBLE SPARSE APPROXIMATION SOLVER ON NEUROMORPHIC HARDWARE

Approved by:

Dr. David Anderson, Advisor
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Christopher Rozell
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Justin Romberg
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Andreas Andreou
Department of Electrical and Computer Engineering
The Johns Hopkins University

Dr. Mark Davenport
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: March 10, 2017

I shake it off, I shake it off.

Taylor Swift

ACKNOWLEDGEMENTS

Behind every successful woman is a tribe of other successful women who have her back. I'm not sure who said this, but there couldn't be a truer statement to convey my experiences as a graduate student. First, my mother, who has uninhibitedly had my back for thirty years. She is always there in moments of celebration but proves to be my biggest champion in circumstances where I or someone else doubts me. Thank you Mom, for leading by example to shape me into a confident, bold woman with a passion to support and instill this confidence in others so that we all succeed. Next, my two closest graduate school friends, Marissa and Kate, who have constantly lifted me up the past few years. I have had countless and invaluable coffee trips with Marissa to celebrate successful meetings, to debrief on the stresses of graduate school as they arise, and to take mental breaks and catch up on how things are going. With Kate being at a different academic institution, we have spent innumerable hours online together discussing so many of the same topics. These women are great friends to me and I will never be able to adequately convey the impact they have had on me. Thank you both for helping me to keep my sanity and for contributing to my success here at Georgia Tech. Last but certainly not least, I have had the privilege to work with many fantastic women from several graduate student organizations over the past three years. My quality of graduate student life improved drastically when I met each one of you. I thank all of you for your leadership, your constant encouragement, and most of all for your friendship.

This isn't to say that I haven't had awesome men in my life that have equally impacted me, because I have. My father raised me to believe that I can do anything and I am forever grateful. He too has supported me in every success and failure, in every confidence and doubt, and I know I would not be the woman I am without his constant love and guidance. My younger brother has also encouraged and celebrated me along the way, and always knows the right or wrong time to say something ridiculous to lighten a mood amidst the

craziness of graduate school. Thank you also to Mark and Michael, who put up with my coming to reading group without having necessarily read the papers and who helped me to shape my work so that I was able to succeed. Thank you to Sebastian, who is a huge supporter of everyone he comes into contact with, for your encouragement and fast friendship. Thank you also to my male peers who have positively impacted my time as a graduate student.

I would also like to thank my advisor and committee members for all of your guidance. Thank you for the time you all spent with me to mold my research pursuits. I have learned so much as a graduate student. Thank you for making this an invaluable experience.

Thank you to my Renovation Church family for your love and support while Jared and I have lived in Atlanta - you have truly left us gloriously ruined for God's transcultural church. I will never be the same and I am glad for it. To my friends that I would consider family in Atlanta: thank you for being here for me and for giving me a reason to absolutely love this city and call it home.

Thank you to my friends from my hometown that have sent me love and encouragement when I needed it most. Thank you also to my extended family - grandparents, aunts, uncles, cousins - for always believing great things for me. And thank you to my Fair family for all of your support and confidence and for giving me seven nieces and a sweet nephew to love in the process.

I save my final piece of gratitude for my husband: thank you for being wholeheartedly with me in every moment, every conversation, and in every victory over the past four years we've spent in graduate school. Thank you for challenging me to be a better person and for encouraging me to pursue even the craziest ideas I may have. Most importantly though, thank you for empowering me and never doubting me for a single second. You are the most incredible, compassionate, selfless man I know and I look forward to this next season in our lives together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This work is motivated by real-world embedded-systems signal processing, which often requires low-power hardware coupled with computationally efficient algorithms. We look to the field of neuromorphic engineering, where advances in biologically inspired algorithms and computing architectures [1] have been developed to achieve computational and power efficient systems.

One such neuromorphic computing architecture is the IBM Neurosynaptic System, also known as the TrueNorth chip [2]. This hardware platform is a low-power, neuromorphic computing architecture with over one million programmable, spiking neurons that operate in parallel in an event-driven manner. The spiking neurons of this hardware platform are representative of how the human brain efficiently represents and processes information. There exists physiological evidence of sparse coding within biological systems to achieve this efficiency [3]. Sparse codes are computed by solving for the sparse approximation of a signal, where the signal is described as a linear combination of a few elements from an overcomplete dictionary.

The sparse approximation problem can be solved using the Locally Competitive Algorithm (LCA) [4], a biologically plausible neural network. Mapping the LCA to the TrueNorth chip lays the framework for processing information in a similarly efficient way to that of the nervous system, offering opportunities for low-power signal reconstruction, signal compression, signal classification, and image enhancement applications [5, 6, 7, 8] in real-world embedded systems. While the TrueNorth chip has shown success in successfully deploying neural networks, the LCA architecture differs from that of deep and convolutional neural networks for which the TrueNorth chip targets due to its recurrent nature and neuronal competition.

We therefore develop a novel design methodology to map the Locally Competitive Algorithm to the TrueNorth chip to solve for the sparse approximation of a signal, offering

the largest LCA dictionaries implemented on neuromorphic hardware to date with perfect precision. We observe low-power consumption in the operation of the LCA on the TrueNorth chip. We also explain methods to map other sparsity-based probabilistic inference problems onto the hardware using our design methodology. We describe the optimal way to achieve high-precision calculations by encoding and decoding signals within time windows. We discuss in detail functional processing units for use on the hardware that offer non-linear thresholds, increased vector-matrix multiplication precision, and the ability to accurately implement a recurrent network on the TrueNorth chip. Our design methodology offers the foundation for low-power embedded systems signal processing applications using the TrueNorth chip.

CHAPTER 1
INTRODUCTION

The concept of neuromorphic systems [9] was introduced over two decades ago, defining such a system as one that is based on the organizing principles of the nervous system. The benefits of neuromorphic systems lie in that biological systems are often-times more effective than technology solutions, especially in terms of computational and power efficiency. As a result, many advances in neuromorphic algorithms and computing architectures [1] have been developed.

The IBM Neurosynaptic System, also known as the TrueNorth chip [2] is a low-power, neuromorphic computing architecture with over one million programmable, spiking neurons that operate in parallel in an event-driven manner. The TrueNorth chip has shown success in implementing a number of neural networks using a high-level, end-to-end ecosystem provided by IBM [10]. Low-level programming is also available to program at the algorithmic level, where the user has access to choose a variety of parameters for every neuron on the chip.

The spiking neurons of the TrueNorth chip are representative of how the human brain efficiently represents and processes information. Physiological evidence exists of sparse coding being employed by biological systems to achieve this efficiency [3]. In sparse coding, redundancy of the environment translates to redundancy of the firing pattern of neurons in response to a stimuli; therefore, the number of neurons responding at any moment is minimized [11, 12]. Sparse codes are computed by solving for the sparse approximation of a signal, where the signal is described as a linear combination of a few elements from an overcomplete dictionary. The dimensionality of the data is not reduced; however, the number of dictionary elements with non-zero values are few relative to the dictionary size. Sparse approximation solvers implemented on low-power hardware offer opportunities for

a variety of signal enhancement and reconstruction applications [6, 7, 8] in real-world embedded systems. An efficient sparse approximation solver implemented on neuromorphic hardware such as the TrueNorth chip lays the framework for processing information in a similarly efficient way to that of the nervous system for signal processing applications.

The Locally Competitive Algorithm (LCA) [4] is a biologically plausible neural network that solves the sparse approximation problem with guarantees to converge to the correct solution [13, 14]. This network has recurrent nature and neurons compete to contribute to the sparse approximation by means of lateral inhibition. While the TrueNorth chip was developed to successfully deploy neural networks, the LCA architecture differs from that of deep and convolutional neural networks for which the TrueNorth end-to-end ecosystem targets. These differences require a novel design methodology to map algorithms with similar architecture to that of the LCA to the TrueNorth hardware. The purpose of our research is therefore to complete the following:

1. Develop a novel design methodology to map high-precision, recurrent networks to low-precision, event-driven hardware

2. Use the above methodology to implement a biologically-plausible sparse approximation solver on neuromorphic hardware

3. Extend the methodology to provide a framework for use in other signal-processing applications

We address the first item in Chapter 3, where we describe computational units we develop that enable us to map high-precision, recurrent architectures to the TrueNorth chip. To achieve the low-power and programmable nature of the TrueNorth chip, design trade-offs were made by developers as they are in any engineering problem. Trade-offs that impact the implementation of recurrent networks on the chip include the following: the chip has a lack of internal memory aside from the constantly evolving neuron potentials, all mathematical operations must be computed at integer precision, and once programmed,

the parameters of neurons are static with no global reset for neuron potentials. We address these hardware constraints and design foundational processing units that we describe in the context of the LCA. We discuss methods to scale the LCA and encode data to work within the integer precision restrictions of the TrueNorth chip. We then provide computational units to perform non-linear thresholds, to emulate on-chip memory for evolving values via recurrence in the system, and to increase the precision of vector-matrix multiplication computations for any sized matrix.

In Chapter 4, we combine the above computational units to implement the largest LCA system on neuromorphic hardware to date. We show that using our methodology the LCA can be exactly implemented on the TrueNorth chip and therefore convergence to the correct sparse approximation is guaranteed. Performance and power results for a variety of system parameters are presented.

We then extend our design methodology to other signal processing applications in Chapter 5. We address solving the least squares, re-weighted ℓ_1, and approximate ℓ_0 problems. We explain the modifications required in detail to successfully implement these systems within our framework. We present preliminary results for the least squares system on the TrueNorth chip.

CHAPTER 2
BACKGROUND

We deliver a design methodology that maps high-precision, recurrent neural networks to low-precision, spiking neuromorphic hardware. We use this methodology to solve signal processing problems on low-power hardware to offer opportunities for use in real-world embedded systems applications.

The IBM Neurosynpatic System, or the TrueNorth chip, has shown success in implementing a number of neural networks using a high-level, end-to-end ecosystem provided by IBM [10]. The Locally Competitive Algorithm (LCA) [4] is a biologically plausible neural network that solves the sparse approximation problem. While the TrueNorth chip was developed to successfully deploy neural networks, the LCA architecture differs from that of deep and convolutional neural networks for which the TrueNorth end-to-end ecosystem targets. In this chapter, we explain the TrueNorth chip and LCA architectures to motivate the need for our novel design methodology.

2.1 The TrueNorth Chip

2.1.1 Hardware Specifications

A highly programmable chip targeting efficient, real-world computing applications is most preferable for this research. The TrueNorth chip [2] meets these needs. The TrueNorth system architecture is useful for a wide variety of neural network applications [2] with efficiency in terms of communication, memory, and computation [15].

The TrueNorth chip is partitioned into a 2D array of cores, creating an on-chip mesh routing network. These cores communicate in an event-driven fashion, sending spike events over the mesh network in the x- then y-directions, followed by a local fan-out to specific

input lines once destination cores are reached. Not only does this increase communication efficiency, but also takes advantage of implicit memory addressing for a memory efficient data structure where input lines are uniquely addressed and therefore implicitly address the outputs from other cores. All events are processed with a synchronization barrier so that all computations are completed for all cores within a single TrueNorth time step, typically operating at one millisecond per step [15]. These time steps are referred to as "ticks".

A single TrueNorth chip contains 4096 neurosynaptic cores, each of which consist of 256 axons (inputs), 256 neurons (outputs), and 256 $\times$ 256 synaptic connections between the two. The TrueNorth chip therefore has millions of brain-inspired programmable neurons equipped with 23 programmable parameters for each neuron, allowing each neuron or group of neurons to take on a wide variety of behaviors [16, 10].

The Neurosynaptic System, 1 million neuron evaluation platform (NS1e) is 125mm$\times$69mm and contains one TrueNorth chip, ARM cores running the Linux operating system, an FPGA performing data conversion and interface translation, as well as various on-board sensors. User interfaces include Gigabit Ethernet, micro USB, I2C, SPI, UART, and spike-based interfaces directly to TrueNorth. By connecting to a small battery and transmitting/receiving data wirelessly, the board has the ability to operate in a full standalone mode, useful in autonomous applications [10].

Multiple NS1e boards can be connected to run networks in parallel or connected in a grid-fashion to execute much larger networks than can be implemented on a single board if more neurons are needed for a particular application. To date, 16 boards have been connected using both methods. The NS1e-16 system connects sixteen boards to run many network instances in parallel. The Neurosynaptic System 16 million neuron evaluation (NS16e) consists of three boards with TrueNorth chips in a 4×4 grid, offering a platform for networks 16 times larger than one NS1e board [10].

2.1.2 End-to-End System Overview

The TrueNorth team offers an end-to-end ecosystem to deploy networks on the TrueNorth chip. This ecosystem consists of a design workflow and a runtime workflow [10]. The design workflow consists of two steps. The first is to prepare a network constrained to the TrueNorth hardware using IBM's Corelet Programming Environment (CPE) [17]. CPE is a MATLAB-based programming suite that supports the object-oriented Corelet Language used to program TrueNorth cores. A corelet is a self-contained module of multiple cores that are programmed to achieve a specific functionality. The second step of the design workflow is to compile a TrueNorth model using programmed corelets to run on IBM's Compass simulator [18] that offers exact 1:1 correspondence to the hardware [15]. At runtime, data is acquired, preprocessed, encoded as spike streams, and run on the TrueNorth chip programmed with the model built during the design workflow. The resulting output spike streams are decoded for analysis to complete the runtime workflow.

High- and low-level frameworks are available to develop networks for use on the TrueNorth chip [10]. Two high-level frameworks are currently offered. The first uses the open-source C++ Caffe framework [19] to train backpropagation-based deep neural networks [20]. The second constrains a deep convolutional neural network to spiking neurons and core-to-core connectivity [21] with customized MATLAB code using functions from MATLAB's MatConvNet library [22]. Both frameworks achieve near state-of-the-art performance [20, 21] and provide the user with abstract frameworks to simplify the configuration process of the TrueNorth chip [10].

The low-level framework requires programming axon and neuron parameters and core-to-core connectivity directly [16]. We detail this framework in the next sub-section.

2.1.3 Composing Networks Using a Low-level Framework

Each TrueNorth core consists of 256 axons (inputs), 256 neurons (outputs), and 256×256 synaptic connections between the two [16]. A core's configuration includes assigning axon

types to each axon, programming 23 parameters and one axon destination for each neuron, and setting the synaptic connections between axons and neurons.

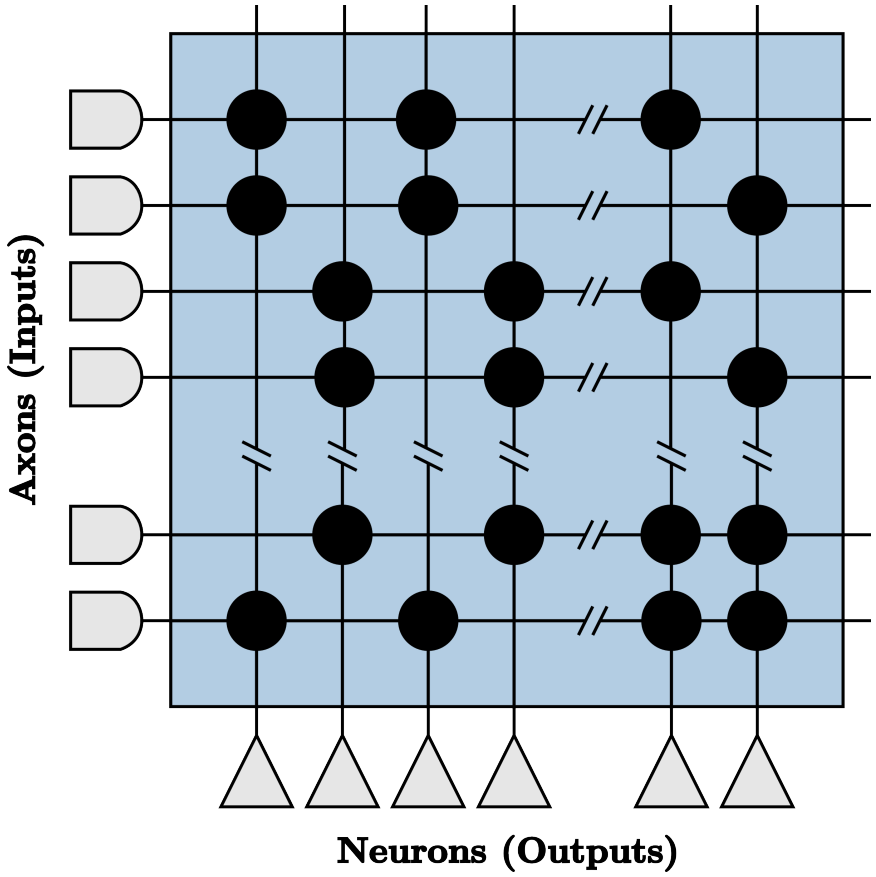

Figure 2.1: A TrueNorth neurosynaptic core.

Each axon i is assigned one type $G_i \in \{0, 1, 2, 3\}$. Each neuron j can assign one integer between -255 and +255 to the four axon types, labeled $s_j^{G_i}$, and can be thought of as synaptic weights if the synaptic connection between the axon and neuron is active. These four assignments can vary for different neurons. Synaptic connections $w_{i,j}$ are binary.

Each neuron also has user-determined positive and negative thresholds α_j and β_j respectively. There are several reset types upon reaching a threshold. We show neuron up-

dates for the two options that apply to our system. The first option is a linear reset:

$$V_m[n+1] = \begin{cases} \text{spike, } V_m[n] - \alpha & \text{if } V_m[n] \geq \alpha \\ V_m[n] + \beta & \text{if } V_m[n] < -\beta \\ V_m[n] & \text{if } -\beta \geq V_m[n] < \alpha \end{cases} \tag{2.1}$$

while a second option is a hard reset:

$$V_m[n+1] = \begin{cases} \text{spike, } V_{\text{rst}} & \text{if } V_m[n] \geq \alpha \\ V_{\text{rst}} & \text{if } V_m[n] < -\beta \\ V_m[n] & \text{if } -\beta \geq V_m[n] < \alpha \end{cases} \tag{2.2}$$

In Figure 2.2, we show an example of a programmed core using 5 of 256 possible axons $i \in \{0:255\}$ and 4 of 256 neurons $j \in \{0:255\}$ connected via a binary synaptic crossbar array $w_{i,j}$. Axons $i = 0, 1, 2, 3, 4$ are types 0, 1, 3, 0 and 2 respectively, shown within the axon symbols in the figure for clarity. The appropriate synaptic weights $w_{i,j} \times s_j^{G_i}$ have been overlaid onto the synaptic connections for clarity. It is important to note that precision is restricted by the limited number of axon types available as shown with axons $i = 0, 3$. As these axons are the same type, any connections between axons $i = 0, 3$ with the same neurons must have the same synaptic weight, such as with neuron $j = 2$ in this example.

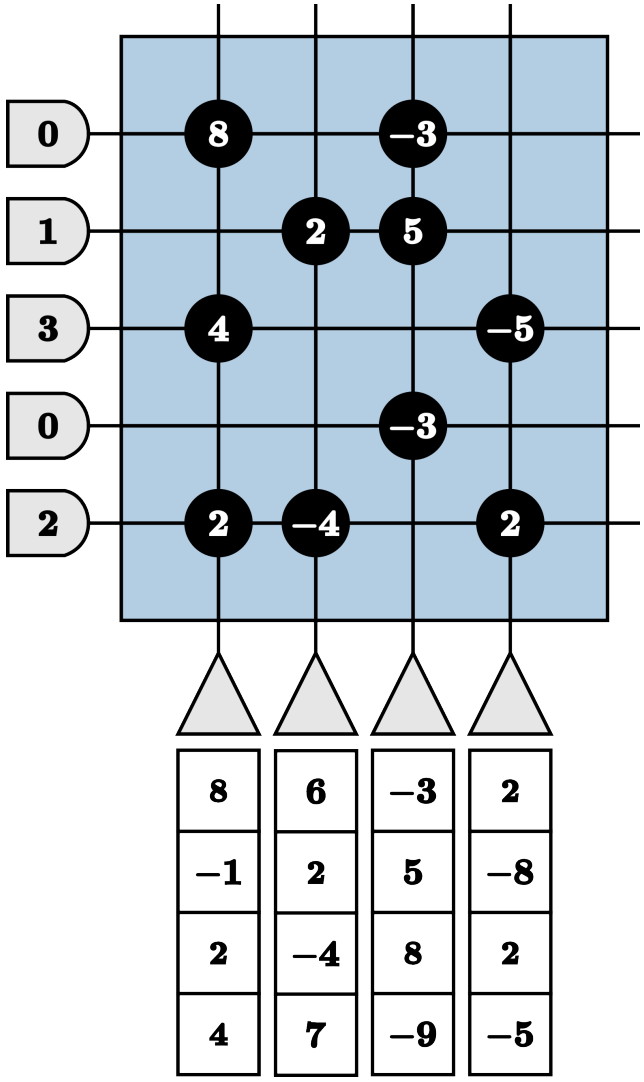

Figure 2.2: An example of a programmed TrueNorth core. Connections and synaptic weights are chosen at random.

Once the chip has been programmed, information is processed through the system using spikes that are routed between cores. Each neuron output can target only one axon input within any core in the system. If a neuron's potential V_j reaches its threshold α_j, it sends a spike to the appropriate axon A_i. For time t, if an axon has received a spike, $A_i(t) = 1$; otherwise, $A_i(t) = 0$. The cores update neuron states simultaneously every 1ms for real-time operation. This clock can be sped up, but doing so can cause spike delivery to be delayed such that simulation results of the system are no longer equivalent to real hardware operation [2, 16]. A simplified version of TrueNorth neuron dynamics for neuron j at time

t is as follows:

$$V_j(t) = V_j(t-1) + \sum_{i=0}^{255} A_i(t) \times w_{i,j} \times s_j^{G_i} \tag{2.3}$$

We show the behavior of TrueNorth neurons using our example core from Figure 2.2 over the course of three TrueNorth ticks in Figures 2.4 to 2.6. We use the simplified TrueNorth neuron dynamics in Equation (2.3) to update the neuron potentials in each tick. We set the positive and negative thresholds for each neuron to $\alpha = 1$ and $\beta = 0$ respective, with a linear reset from Equation (2.1). We assume the initial potential $V(0)$ for each neuron is zero. Updated neuron potentials for each tick are shown within the neuron symbols. We denote inputs by red dots to the left of the axons. Inputs reach the axons in order from right to left. For example, given the above notation $A_0(1) = 1$ and $A_3(2) = 1$. Output spikes are shown as red dots exiting the neurons. In Figure 2.3, we show the system before any input spikes are received.

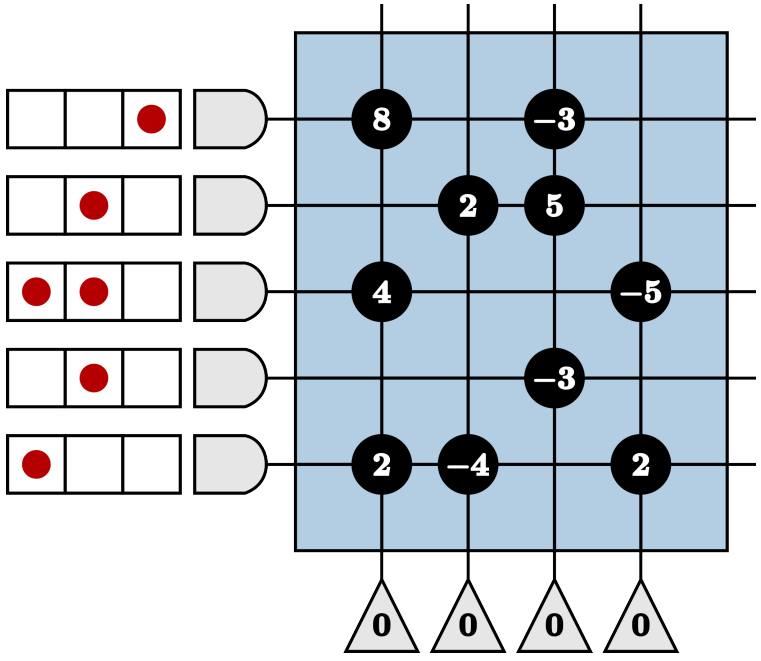

Figure 2.3: An example of a programmed TrueNorth neurosynaptic core with inputs for three ticks.

During the first tick in Figure 2.4, only axon $i = 0$ receives an input spike. This

spike impacts the neuron potentials of only the neurons that are connected to the axon $i = 0$ by synaptic connections. These connections have synaptic weights overlaid onto the connections in the figure and the neuron potentials are updated and shown inside of the neuron symbol to reflect the new potentials. Although two neurons exceed the threshold in the first tick, the output spikes do not emit until the next time step, tick $= 2$.

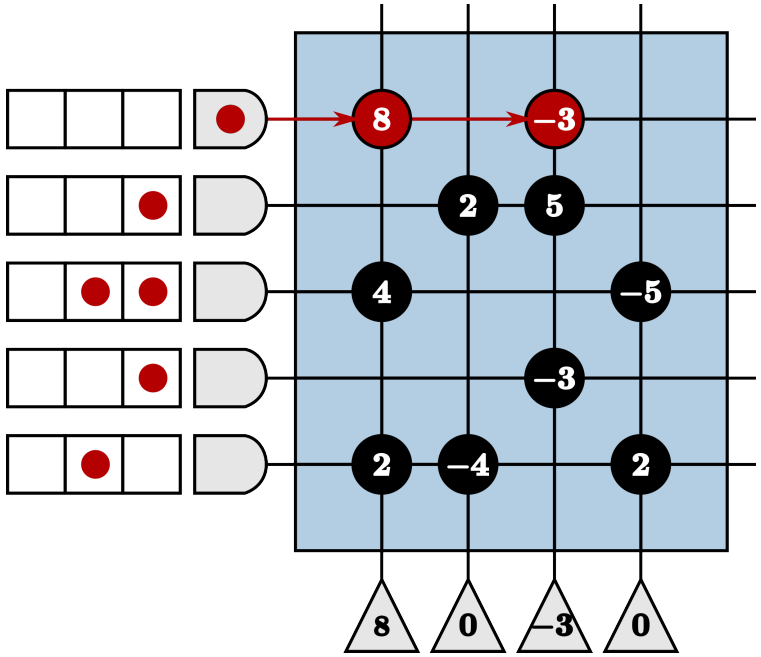

Figure 2.4: TrueNorth neuronal behavior for tick one of three.

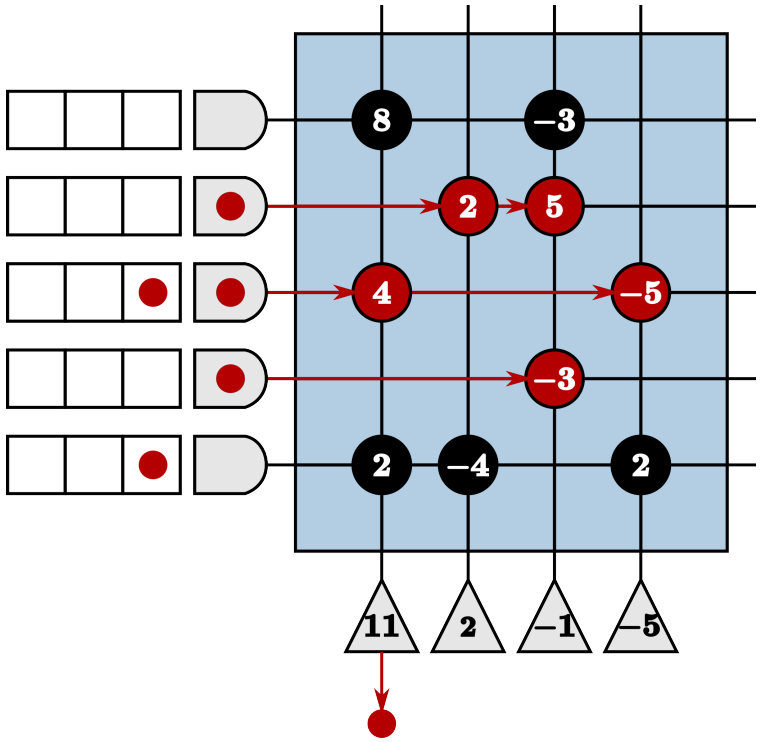

Figure 2.5: TrueNorth neuronal behavior for tick two of three.

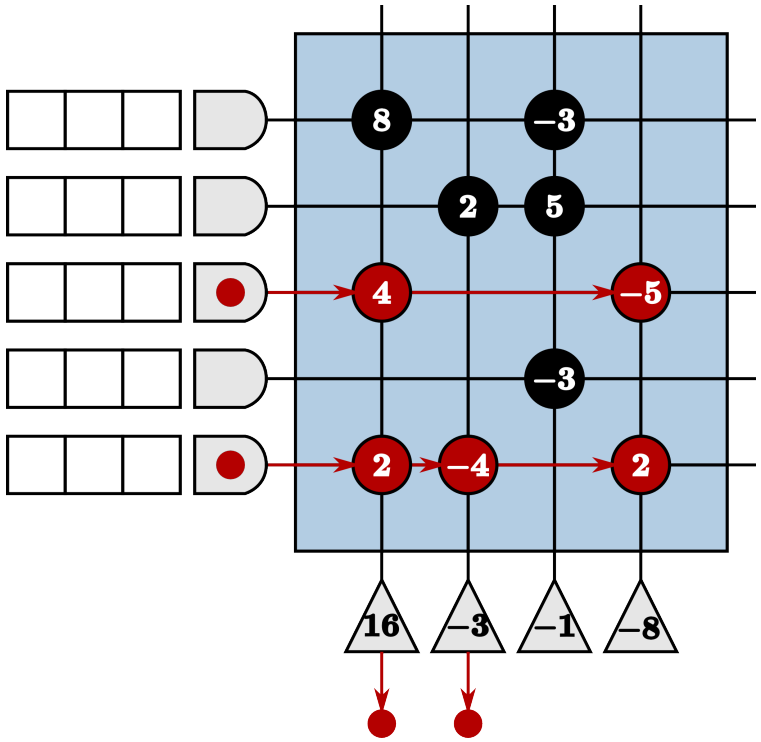

Figure 2.6: TrueNorth neuronal behavior for tick three of three.

2.2 The Sparse Approximation

An *N*-pixel image $y \in R^N$ can be represented as a linear combination of a set of *M* basis vectors that span the space of R^N. Each basis vector is a column in $\mathbf{\Psi}$ and the weights of each basis vector are contained in a length *M* vector $\mathbf{a}$. The following equation considers noiseless, perfect reconstruction of the signal [11, 23]:

$$\mathbf{y} = \sum_{m=1}^{M} a_m \phi_m = \Psi \mathbf{a} \tag{2.4}$$

Much of natural image information however, is contained in only a fraction of the full resolution signal. Given a set of data vectors where $M > N$, also referred to as an over-complete dictionary, a natural image can be decomposed into a linear combination of very few of the vectors in the dictionary, referred to as sparse coding [24]. Sparse coding relies on the redundancy of the visual environment and the redundancy of responses to the dictionary vectors. Dimensionality is not reduced in this model, but rather the goal is to have the fewest non-zero entries as possible in $\mathbf{a}$ [11]. These representations can transform complicated natural data to a simpler, more explicit form [3], requiring less processing power and memory throughout a system.

Finding $\mathbf{a}$ in sparse coding is referred to as solving the sparse approximation problem. Given the signal $\mathbf{y}$, its sparse approximation $\mathbf{a}$ can be computed relative to a dictionary Φ by solving the following problem:

$$\min_{\mathbf{a}} \|\mathbf{a}\|_0 \text{ such that } \mathbf{y} = \Phi \mathbf{a}. \tag{2.5}$$

The ℓ_0 constraint makes this problem very difficult, but a solution to this problem can be approximated with greater efficiency by substituting the ℓ_0 constraint with the ℓ_1 norm [25]. When signal noise is taken into account however, a trade-off λ must be considered between sparsity and reconstruction accuracy. This yields the solvable, unconstrained optimization problem where the first term places emphasis on accurate reconstruction of the original

signal while the second ensures sparsity in the sparse approximation [26]:

$$\underset{\mathbf{a}}{\operatorname{argmin}} \left(\frac{1}{2} \left\| \mathbf{y} - \Phi \mathbf{a} \right\|_2^2 + \lambda \left\| \mathbf{a} \right\|_1 \right) \tag{2.6}$$

2.3 The Locally Competitive Algorithm

The Locally Competitive Algorithm (LCA) [4] approaches Equation (2.6) in a biologically-inspired manner as a low-power, continuous-time sparse approximation solver. The LCA is a neural network that guarantees convergence to the correct solution [13, 14]. This network consists of "neurons" or "nodes" that compete to contribute to the sparse approximation of the original signal by lateral inhibition.

Each LCA node represents an element from an overcomplete dictionary Φ combined to reconstruct a signal $\mathbf{y}(t)$ via its sparse approximation $\mathbf{a}(t)$: $\hat{\mathbf{y}}(t) = \Phi \mathbf{a}(t)$. The internal state of each node is contained in the vector $\mathbf{u}(t)$. Each node's state changes according to the dynamics in Equation (2.7).

$$\dot{u}_m(t) = \frac{1}{\tau} \left(b_m(t) - u_m(t) - \sum_{m \neq n} G_{m,n} a_n(t) \right) \tag{2.7}$$

where τ is a system-determined time constant. The initial projection of an LCA node $b_m(t)$ is calculated by $b_m(t) = \langle \Phi_m, y_m(t) \rangle$. The matrix G performs the lateral inhibition of nodes by $a_m G_{m,n}$, where $G_{m,n}$ is calculated by taking the inner product of each node with all other nodes $G_{m,n} = \langle \Phi_m, \Phi_n \rangle$. Larger values within this matrix signify more closely related nodes. Active nodes suppress nodes based on the values found in G to reduce redundancy in the sparse approximation of a signal.

To determine whether a node is active, meaning it contributes to the sparse approximation of the signal [4], a soft threshold function is used. If the state of a node calculated by Equation (2.7) exceeds a threshold λ, the node becomes active and contributes to the sparse approximation while suppressing other similar node activity. The threshold λ serves as the

tradeoff between reconstruction of the original signal and sparsity of the sparse approximation as discussed for Equation (2.6). The soft threshold is computed using Equation (2.8).

$$a_m(t) = T_\lambda(u_m(t)) = \begin{cases} \frac{u_m(t)}{g_m} - \text{sign}(u_m(t))\frac{\lambda}{g_m} & \text{if } |u_m(t)| \geq \lambda \\ 0 & \text{if } |u_m(t)| < \lambda \end{cases} \tag{2.8}$$

The term g_m consists of the m diagonal values of $\Phi^T\Phi$ and is used for nodes with non-uniform norms. For a dictionary with unit norm nodes, $g_m = 1$ and the term can be disregarded in Equation (2.8). A plot of the soft threshold outputs for a unit norm node and $\lambda = 1$ is shown in Figure 2.7.

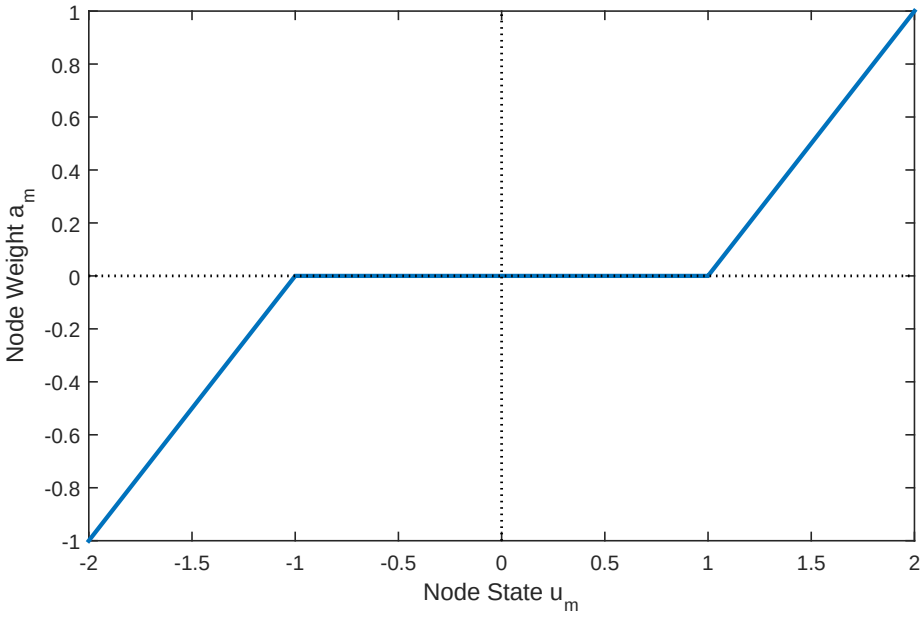

Figure 2.7: The soft threshold function.

The resultant interacting node dynamics of an LCA system are shown in Figure 2.8. The soft threshold is denoted by the dashed, horizontal lines in the plot. As the node states of active LCA nodes reach the soft threshold, similar nodes are suppressed below the threshold and therefore do not contribute to the sparse approximation of the original signal.

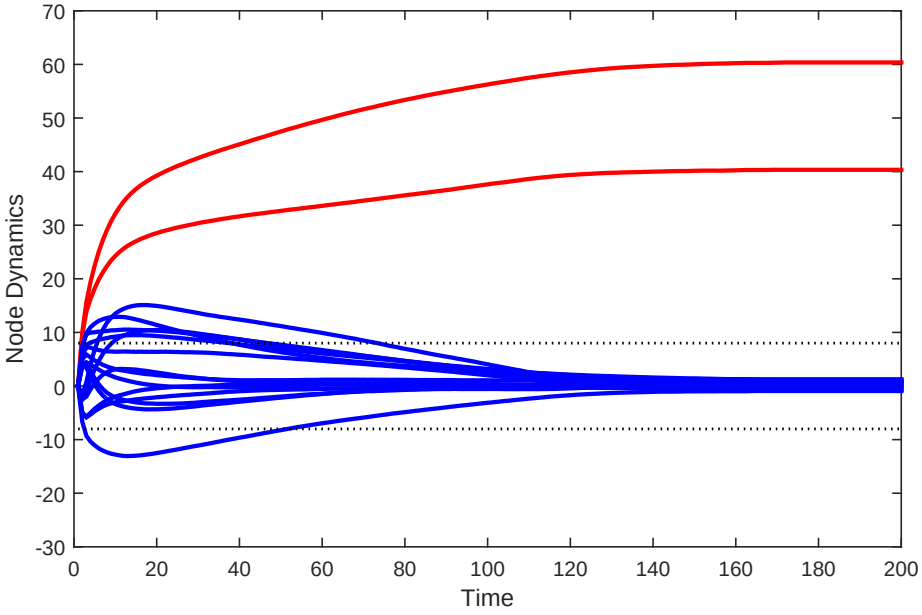

Figure 2.8: The interacting node dynamics of an LCA system.

The sparse approximation of the above LCA system is shown in Figure 2.9. The dashed lines again represent the soft threshold. Node weights that exceed the soft threshold are those used in the accurate reconstruction of the signal.

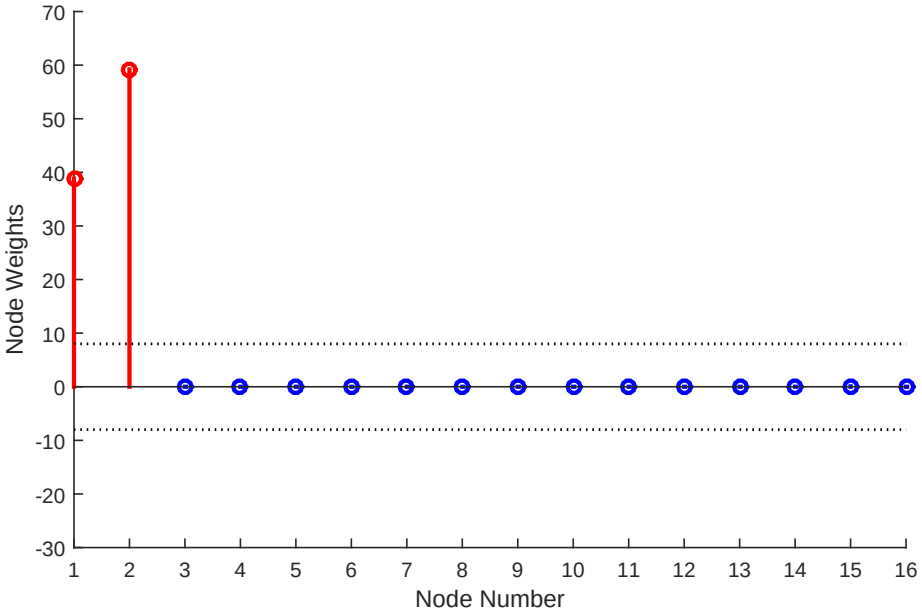

Figure 2.9: A sparse approximation of a signal computed by the LCA.

CHAPTER 3
DESIGN METHODOLOGY TO MAP HIGH-PRECISION, RECURRENT ALGORITHMS TO TRUENORTH

A novel design methodology is required to map high-precision, recurrent architectures to the TrueNorth chip. To achieve the low-power and programmable nature of the TrueNorth chip, design trade-offs were made. Trade-offs that impact the implementation of recurrent networks on the chip include the following: the chip has a lack of internal memory aside from the constantly evolving neuron potentials, all mathematical operations must be computed at integer precision, and once programmed, the parameters of neurons are static with no global reset for neuron potentials.

The TrueNorth chip however, offers millions of programmable, spiking neurons, each with a number of parameters that allow each neuron or a small group of neurons to take on a wide variety of behaviors [27, 16]. We cleverly exploit the properties of TrueNorth neurons to create three foundational processing units to work within these hardware constraints when programming at the low-level on the TrueNorth chip. These units perform non-linear thresholds, emulate on-chip memory for evolving values via recurrence in the system, and increase the precision of vector-matrix multiplication computations for any sized matrix. To describe these units, we detail our methods in the context of the Locally Competitive Algorithm (LCA). We first scale the LCA and appropriately encode data to work within the integer precision restrictions of the TrueNorth chip. We then break the scaled LCA node dynamics into sub-functions and detail our computational units according to this system.

3.1 Scaled LCA Dynamics for the TrueNorth Chip

The LCA node dynamics can not be directly implemented using the TrueNorth neuron dynamics. To implement the LCA within the neuronal properties of the TrueNorth chip, we

discretize the LCA algorithm by simple time-domain sampling for which node dynamics are shown in Equations (3.1) to (3.3). The τ parameter becomes a learning rate in the discrete system.

$$\mathbf{u}[n+1] = \mathbf{u}[n] + \Delta\mathbf{u}[n] \tag{3.1}$$

$$\Delta\mathbf{u}[n] = \frac{1}{\tau}\left(\mathbf{b} - \mathbf{u}[n] - G\mathbf{a}[n]\right) \tag{3.2}$$

$$a_m[n] = T_\lambda(u_m[n]) = \begin{cases} \frac{u_m[n]}{g_m} - \text{sign}(u_m[n])\frac{\lambda}{g_m} & \text{if } |u_m[n]| \geq \lambda \\ 0 & \text{if } |u_m[n]| < \lambda \end{cases} \tag{3.3}$$

Given a large τ, one expects a more precise computation of a signal's sparse approximation. A consequence however, will be much smaller values throughout the LCA dynamics that the TrueNorth chip can not accommodate. We therefore scale our system by τ^2 shown in Equations (3.4) to (3.6) such that all values are greater than or equal to one and can be accurately represented within the TrueNorth integer precision restrictions.

$$\tau^2\mathbf{u}[n+1] = \tau^2\mathbf{u}[n] + \tau^2\Delta\mathbf{u}[n] \tag{3.4}$$

$$\tau^2\Delta\mathbf{u}[n] = \tau\mathbf{b} - \tau\mathbf{u}[n] - G\tau\mathbf{a}[n] \tag{3.5}$$

$$\tau a_m[n] = T_{\tau\lambda}(\tau u_m[n]) = \begin{cases} \frac{\tau u_m[n]}{g_m} - \text{sign}(u_m[n])\frac{\tau\lambda}{g_m} & \text{if } |\tau u_m[n]| \geq \tau\lambda \\ 0 & \text{if } |\tau u_m[n]| < \tau\lambda \end{cases} \tag{3.6}$$

3.2 Data Encoding

In recurrent algorithms, values are constantly evolving. To map such changing values onto the TrueNorth chip where neuron parameters are static, we must encode data to achieve high precision. In the context of LCA, values $\mathbf{u}, \Delta\mathbf{u}$, and $\mathbf{a}$ are constantly evolving until the LCA system converges. This prevents us from using the programmable synaptic weights offered by the TrueNorth chip directly. For instance, suppose we are performing the summation in Equation (3.5). Ideally, for the LCA node m we would connect three TrueNorth axons to the same TrueNorth neuron, one axon to represent $\tau u_m[n]$, one to represent $\tau b_m[n]$, and the last to represent $\tau G a_m[n]$. We would set the synaptic weights of the axons to be $-\tau u_m[n]$, $\tau b_m[n]$, and $-\tau G a_m[n]$ respectively. The output spikes of the neuron would therefore represent the solution $\tau^2 \Delta u_m[n]$. However, two terms in the summation are constantly evolving as the system converges, whereas the synaptic weights cannot be changed once the chip has been programmed.

We illustrate this example with the aforementioned ideal programming scenario in Figure 3.1 and two subsequent iterations using this scenario for incorrect computation in Figures 3.2 and 3.3. In Figure 3.2, the synaptic weights are programmed to match that of the values $-\tau u_m[n] = -5$, $\tau b_m[n] = 10$, and $-\tau G a_m[n] = -3$. By sending one input spike to each axon and setting a positive threshold $\alpha = 1$ and linear reset $V(t) = V(t-1) - \alpha$, we see two output spikes from the neuron, accurately computing $\tau b_m[1] - \tau u_m[1] - \tau G a_m[1] = 10 - 5 - 3 = 2$. However in the next iteration, $\tau u_m[1] \neq \tau u_m[2]$ and $\tau G a_m[1] \neq \tau G a_m[2]$, causing the system to incorrectly compute $\tau^2 \Delta u_m[2]$ since the synaptic weights are static parameters.

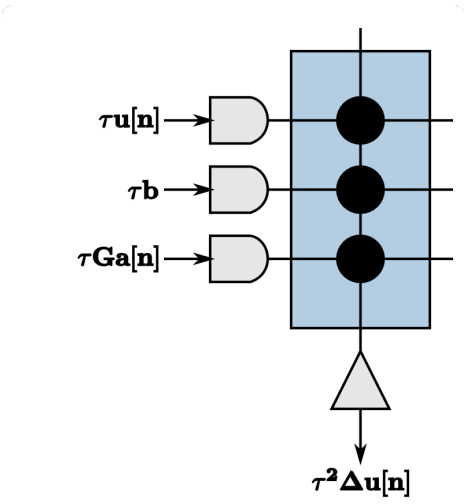

Figure 3.1: The ideal programming scenario for summation on the TrueNorth Chip

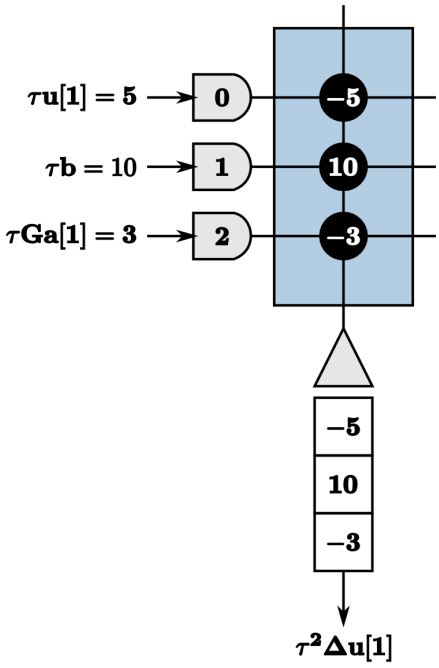

Figure 3.2: The first iteration computes the summation accurately using the ideal programming scenario.

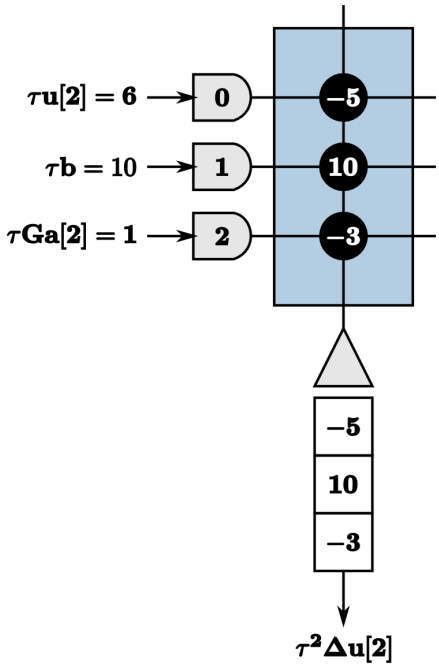

Figure 3.3: The second iteration computes the summation incorrectly using the ideal programming scenario.

We therefore encode values using a time window for each LCA iteration. The value of an LCA variable at each iteration is determined by counting the number of spikes within the given time window. Spikes can occur anywhere within the window to contribute to the resultant value. We use a window of w TrueNorth ticks for each LCA iteration, where w is greater than or equal to the largest value you would expect to see in a system. We show the same example above in Figures 3.4 and 3.5 with correct encoding techniques and window lengths $w = 10$.

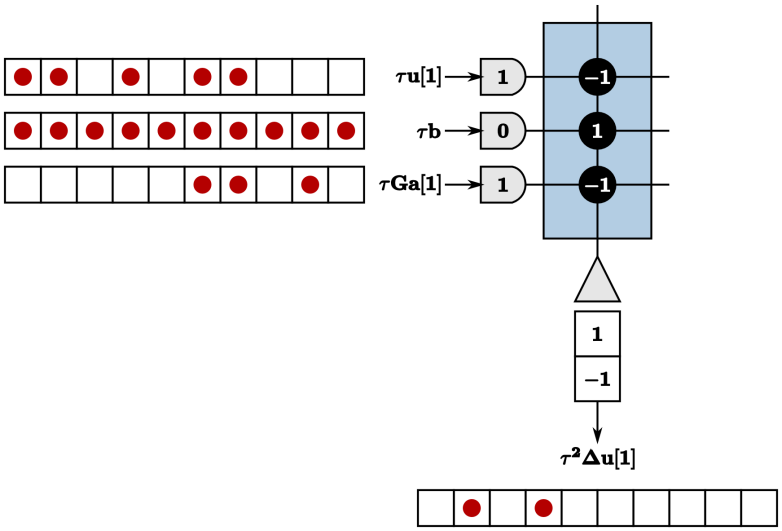

Figure 3.4: The first iteration computes the summation accurately using the correct data encoding scheme to accommodate recurrent input spikes.

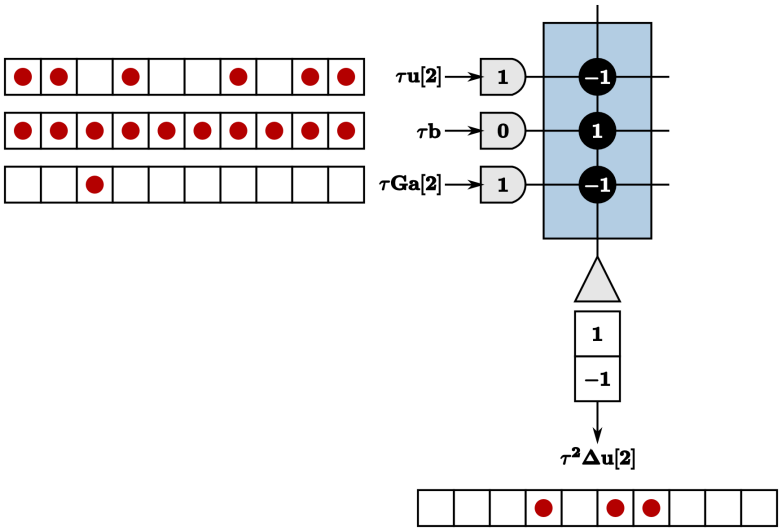

Figure 3.5: The second iteration also computes the summation accurately using the correct data encoding scheme to accommodate recurrent input spikes.

If a window is chosen with too few ticks to accommodate the largest anticipated value, we saturate TrueNorth neurons and incorrectly compute values. In the context of LCA, we plot the node dynamics $\mathbf{u}$ as calculated by TrueNorth with insufficient window sizes compared to the correct node dynamics calculated by a discrete LCA system. The node dynamics do not match demonstrating incorrect computations, shown in Figure 3.6.

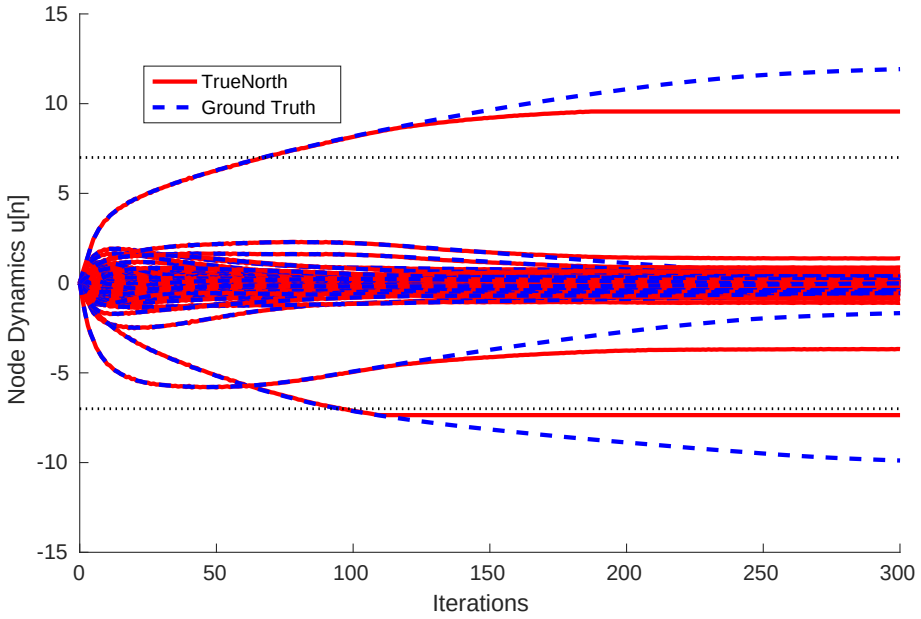

Figure 3.6: Node dynamics calculated by the TrueNorth chip versus those by a discrete LCA system. The window size chosen for this example is too small to accurately compute the sparse approximation.

Given that we scale recurrent systems to achieve high precision, we oftentimes need large windows to accurately compute variables. As it stands, this is not ideal for applications requiring real-time or near real-time calculations. One might speculate that future hardware iterations or hardware tailored to this type of high-precision encoding will offer clock cycles faster than the 1ms the TrueNorth chip offers. Consequently, we choose to focus on correct computations rather than the duration needed to compute values.

3.3 Processing Units

To describe these units, we detail our methods in the context of the Locally Competitive Algorithm (LCA). We break the scaled LCA node dynamics into sub-functions denoted by squares in Figure 3.7 and walk through programming each using our processing units.

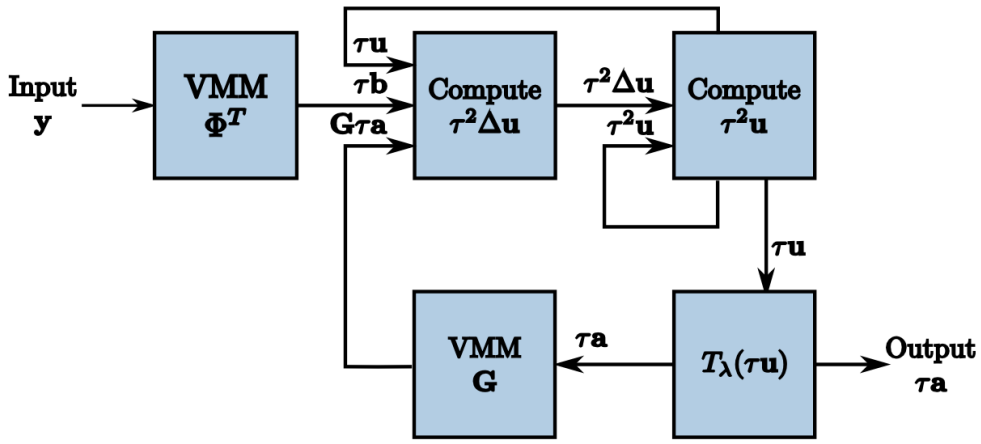

Figure 3.7: The scaled LCA broken into sub-functions.

3.3.1 Programming Framework

In the prior section we touched upon how to program the sub-function labeled "Compute $\tau^2\Delta$u". The summation is only computed correctly if the incoming and outgoing spikes are consistently the same sign. We do not want to limit all values to either strictly positive or strictly negative, therefore we represent the values in our system using positive and negative neurons without any advanced information. TrueNorth neurons can not emit negative spikes, and while biologically plausible in the context of biological systems' neurons, this requires TrueNorth neurons to be repeated so that some neuron outputs represent the standard positive spikes while others represent negative values in the LCA system. For instance, if the true value of the initial projection $b_m = \Phi^T y_m$ is positive, we expect output spikes from the neuron that represents the positive values of b_m. If the true value of b_m is negative, we instead expect output spikes from the neuron that represents negative values.

For this technique to work, the TrueNorth neuron potentials must always be equivalent with opposite signs for the positive and negative representations of b_m. For instance, if $b_m = -2$ and our thresholds are one, the positive representation will have a neuron poten-

tial of $V_m^+ = -2$ while the negative representation has a neuron potential of $V_m^- = +2$. Therefore, the negative representation neuron emits two spikes to output a negative value for b_m. To deploy this technique, the signs of the synaptic weights of the "negative neurons" are reversed relative to the "positive neurons", Figure 3.8.

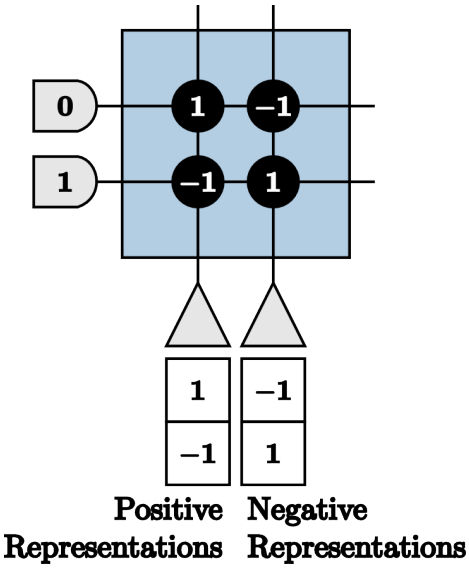

Figure 3.8: A core with neurons repeated and synaptic weights reversed to represent the positive and negative values of the output.

For now, to motivate the methods used to represent positive and negative values, we assume TrueNorth neuron potentials are reset by

$$V_m[n+1] = \begin{cases} \text{spike, } V_m[n] - \alpha & \text{if } V_m[n] \geq \alpha \\ V_m[n] + \beta & \text{if } V_m[n] \leq -\beta \\ V_m[n] & \text{if } -\beta < V_m[n] < \alpha \end{cases} \tag{3.7}$$

where α is the positive threshold and β is the negative threshold for the neuron. It is important to note that upon reaching the negative threshold, a neuron potential resets without the neuron emitting spikes. We choose $\alpha, \beta = 1$ to see the effects of the chosen parameters using example neurons in Figure 3.9. These neurons are the positive and negative representations of the first LCA node, the first being the positive. The axons receive the input spikes denoted by red circles. Incoming spikes reach the axons in order from right to left.

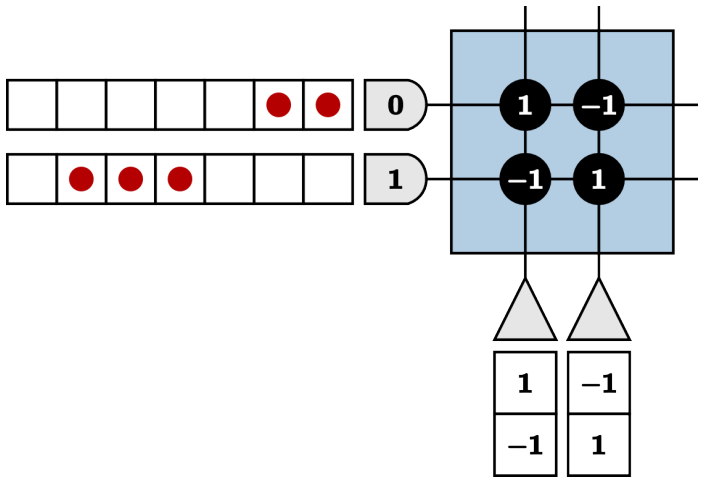

Figure 3.9: The positive and negative representations of a value with inputs. Neuron potentials and outputs are analyzed in Table 3.1.

Table 3.1 shows the neuron potentials of the neurons V^+ and V^- and the outputs of the neurons Out$^+$ and Out$^-$ for a window of $w = 7$ ticks. These calculations use the neuron resets in Equation (3.7). The total output of the negative representation neuron in w ticks is subtracted from the total output of the positive representation neuron in the same w ticks to find the final value. In this example we are multiplying [2 3], shown by the encoded inputs, by [1 -1]^T, shown by the synaptic weights of the first neuron. We are essentially performing subtraction $2 - 3 = -1$ by sending two input spikes with weights of positive one and three input spikes with weights of negative one. We see two output spikes from the positive representation neuron and three from the negative and therefore calculate the final value correctly.

This programming technique is promising; however, the actual TrueNorth neuron resets have asymmetric thresholds and pose a complication. Rather than membrane potentials resetting upon the potential being less than or equal to the negative threshold, they only reset once they exceed the negative threshold, while the positive thresholds reset as we show in Equation (3.7). The thresholds in the TrueNorth chip actually have the following

Table 3.1: TrueNorth neuron states and outputs of the positive and negative representation neurons with reset assumptions from Equation (3.7).

Tick	V^+	V^-	Out$^+$	Out$^-$
1	1	-1	0	0
2	1	-1	1	0
3	0	0	1	0
4	-1	1	0	0
5	-1	1	0	1
6	-1	1	0	1
7	0	0	0	1

neuronal behavior:

$$V_m[n+1] = \begin{cases} \text{spike, } V_m[n] - \alpha & \text{if } V_m[n] \geq \alpha \\ V_m[n] + \beta & \text{if } V_m[n] < -\beta \\ V_m[n] & \text{if } -\beta \geq V_m[n] < \alpha \end{cases} \tag{3.8}$$

As a result, the actual TrueNorth thresholds do not produce identical neuron states with opposite signs for positive and negative representations as we previously assume. This produces states and outputs from the same example shown in Figure 3.9, with an incorrect final value of $2 - 2 = 0$. We show these values in Table 3.2.

Table 3.2: Actual TrueNorth positive and negative representation neuron states and outputs with resets from Equation (3.8).

Tick	V^+	V^-	Out$^+$	Out$^-$
1	1	-1	0	0
2	1	-2	1	0
3	0	-1	1	0
4	-1	0	0	0
5	-2	1	0	0
6	-2	1	0	1
7	-1	0	0	1

For values to take on different polarities over time on the TrueNorth chip, the positive and negative thresholds need to be symmetric, meaning they both must reset by our as-

sumptions in Equation (3.7) where resets occur upon meeting either threshold. Therefore, we develop a way to offset the negative thresholds to achieve this property. We start by removing the negative threshold parameter. We then repeat all neurons, since TrueNorth neurons are restricted to only one destination axon, and send the outputs back to the same core. Any output from the positive representation of a neuron is sent back to its respective negative representation neuron and vice versa. This requires two times the original number of neurons in the system and additional axons to accommodate the feedback. We extend our example in Figure 3.10 to correct for the asymmetric thresholds using this method. We complete the figure by showing the spikes from repeated neurons for feedback in the appropriate ticks within the input window. In doing so, at tick $= 2$ the negative representations with $V^- = -1$ also receive an input spike of $+1$, resetting the state to $V^- = V^+ = 0$. This technique yields states and outputs equivalent to those in Table 3.1.

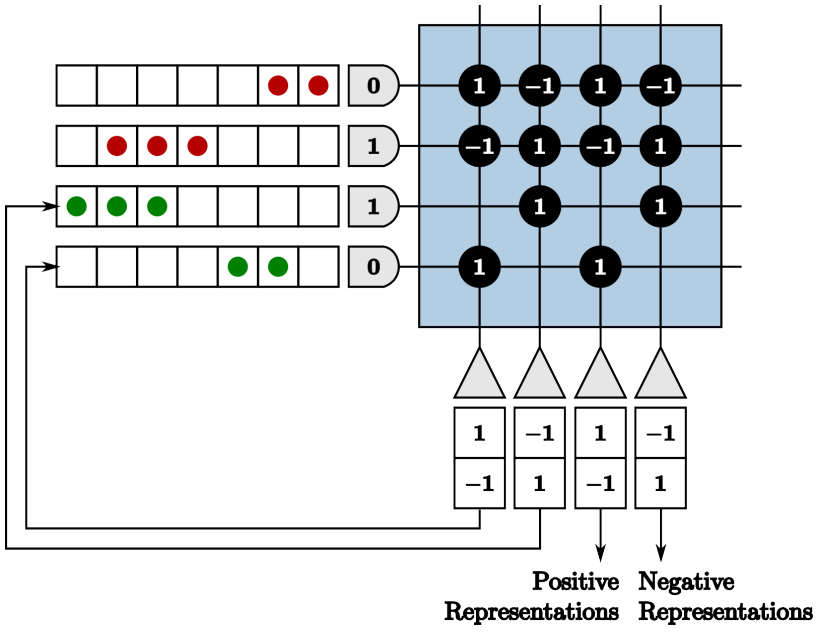

Figure 3.10: Neurons repeated and outputs sent back to respective neurons to result in symmetric thresholds.

Using this technique, we can now represent positive and negative outputs via neurons. However, this only works if we have inputs that reflect strictly positive or strictly negative values. We again do not want to place this kind of restriction on our system. Therefore,

we also break our inputs into both positive and negative representations. This process is similar to having positive and negative representations for neurons: we repeat axons with opposite signs for synaptic weights.

We show an example of repeated axons in Figure 3.11. Positive inputs are represented by the first two axons and negative inputs by the second two axons. For this example, the input would be [2 -3] multiplied as before by [1 -1] resulting in output $2 + 3 = 5$. The remaining axons are designated for feedback to create symmetric thresholds. The first two neurons are used for feedback, positive and negative representations respectively and the outputs of these neurons are displayed in the appropriate axon destination tick locations. The remaining neurons serve as the the positive and negative representations of the correctly calculated value for output of the core.

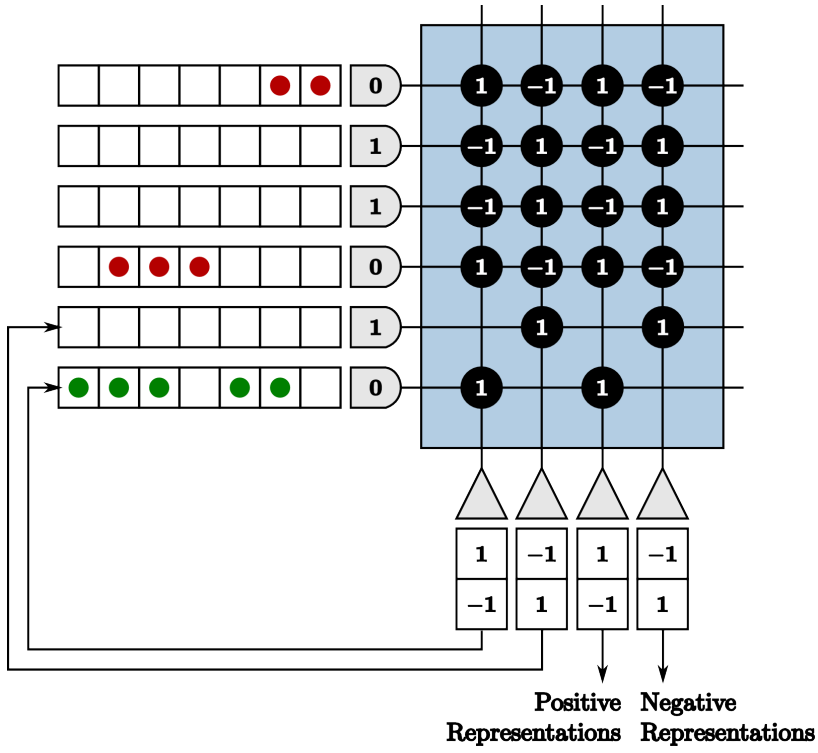

Figure 3.11: A core with repeated axons and neurons to accommodate positive and negative inputs and outputs.

Each of our computational units must be programmed with positive and negative representations for every axon and neuron. Each computational unit must also be programmed

with the appropriate feedback of the neuron outputs to ensure that the positive and negative representations' neuron states are the same every clock cycle. We explain our computational units assuming all incoming and outgoing values are positive to adequately convey programming techniques. However, the reader should note that far more resources are used on the TrueNorth chip to accommodate sign and feedback requirements.

3.3.2 Non-linear Threshold

We describe our non-linear threshold processing unit using the sub-function labeled $T_{\tau\lambda}(u)$ in Figure 3.7. This sub-function performs the non-linear soft threshold in Equation (3.6). The parameter λ serves as the trade-off between reconstruction error and sparsity of the sparse approximation of a signal. We do not want to require any prior information about user-determined parameters in advance, thus we add the threshold $\tau\lambda$ as a user input to our system. Inputs to our soft threshold core are ideally $\tau\mathbf{u}/\text{diag}(G)$ and $\tau\lambda/\text{diag}(G)$ with synaptic weights of one and negative one respectively in Figure 3.12.

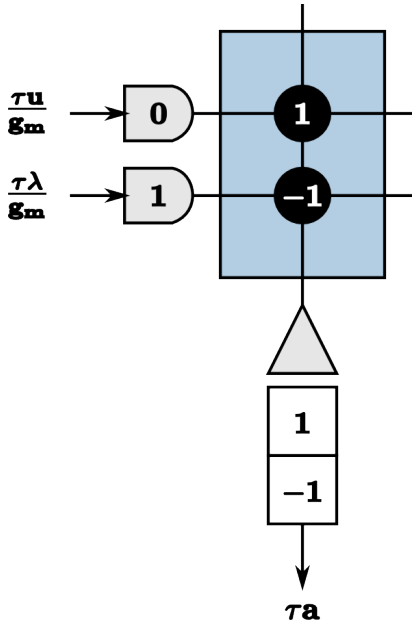

Figure 3.12: An example core to perform the non-linear soft threshold for one LCA node. Only the positive representations of inputs and outputs are shown.

The soft threshold sets any LCA nodes in $|\tau\mathbf{u}/\text{diag}(G)| < \tau\lambda$ to zero while otherwise

adding or subtracting $\tau\lambda$ from $-\tau\mathbf{u}/\text{diag}(G)$ or $\tau\mathbf{u}/\text{diag}(G)$ respectively. For instances where $|\tau\mathbf{u}/\text{diag}(G)|$ exceeds the soft threshold, using the TrueNorth linear neuron potential resets

$$V_m[n+1] = \begin{cases} \text{spike, } V_m[n] - 1 & \text{if } V_m[n] \geq 1 \\ V_m[n] & \text{if } V_m[n] < \alpha \end{cases} \tag{3.9}$$

we compute the soft threshold correctly shown in Figure 3.13. In this example, $\tau u_m/g_m = 5$ and $\tau\lambda = 4$, therefore $\tau a_m = 1$, denoted by solid red circles in the encoded windows.

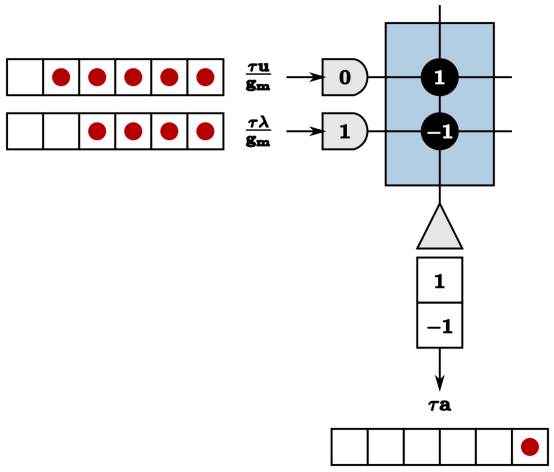

Figure 3.13: The soft threshold performed for one LCA node when $\tau\mathbf{u}/\text{diag}(G)$ exceeds the threshold.

However, for values that fall below the soft threshold, we see the following:

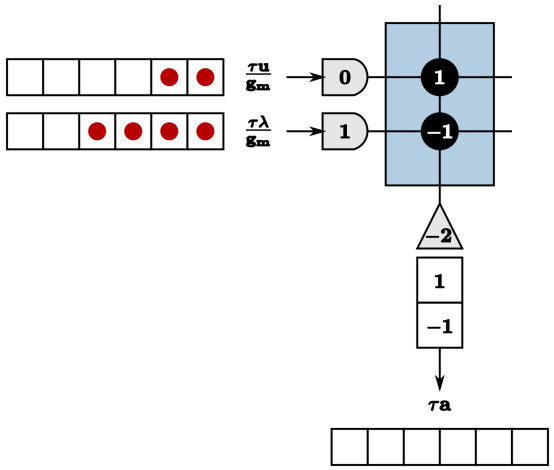

Figure 3.14: The soft threshold performed incorrectly for one LCA node when $\tau\mathbf{u}/\text{diag}(G)$ falls below the threshold.

At the end of a time window length w, the negative neuron potential negative two holds for the next iteration, causing the following iteration's soft threshold to be performed incorrectly. For instance, in the next iteration if we see $\tau u_m/g_m$ exceeds the threshold as it did in Figure 3.13, we would see the updated neuron potential $-2+5-4=-1$, and therefore would not see any output spikes. This would calculate $\tau a_m = 0$ rather than the correct $\tau a_m = 1$.

As a result, we set positive and negative thresholds to be one and zero respectively, and instead choose a hard reset of zero for neuron potentials:

$$V_m[n+1] = \begin{cases} \text{spike, } 0 & \text{if } V_m[n] \geq 1 \\ 0 & \text{if } V_m[n] < 0 \\ V_m[n] & \text{if } 0 \geq V_m[n] < 1 \end{cases} \tag{3.10}$$

We guarantee that we never receive an increase in neuron potential greater than one for a single tick in this core, therefore the hard reset upon reaching $\alpha = 1$ does not negatively impact the calculation. Any neuron potentials that fall below zero are reset to zero thus the next iteration can accurately compute the soft threshold with an initial neuron potential of zero.

To send $\tau\mathbf{u}/\text{diag}(G)$ into the non-linear threshold unit, we must set the prior core's threshold to $\alpha = \tau\text{diag}(G)$ on the neurons that compute $\tau^2\mathbf{u}$ to for the outputs to result in $\frac{\tau\mathbf{u}}{\text{diag}(G)}\frac{1}{\tau\text{diag}(G)} = \frac{\tau\mathbf{u}}{\text{diag}(G)}$. Due to the thresholds being greater than one, we risk the misalignment of spikes due to residual potential between LCA iterations. For instance, if we use thresholds of $\tau\text{diag}(G)$ in the core that computes $\tau^2\mathbf{u}$, unless we have an output value that is an exact multiple of $\tau\text{diag}(G)$, we will have neuron potentials remaining between iterations. When this occurs, future iterations' spikes do not align with the user generated input spikes $-\tau\lambda/\text{diag}(G)$ as they now spike at unpredictable times. Therefore, we scale the soft threshold function by $\tau\text{diag}(G)$ so that the neurons that compute $\tau^2\mathbf{u}$ are $\alpha = 1$, resulting in the following soft threshold $T_{\tau^2\lambda}(\tau^2\mathbf{u})$ implemented in our corelet:

$$\tau^2 g_m a_m[n] = \begin{cases} \tau^2 u_m[n] - \text{sign}(u_m[n])\tau^2\lambda & \text{if } |\tau^2 g_m u_m[n]| \geq \tau^2\lambda \\ 0 & \text{if } |\tau^2 g_m u_m[n]| < \tau^2\lambda \end{cases} \tag{3.11}$$

Inputs to the core are now $\tau^2\mathbf{u}$ and a series of $\tau^2\lambda$ user-generated spikes with synaptic weights of $+1$ and -1 respectively. The first spikes for each input align in the first tick of a time window of w ticks. We choose positive thresholds $\alpha = \tau\text{diag}(G)$ and negative thresholds $\beta = 0$ with the hard reset above. Any LCA node states that fall within $|\tau^2\text{diag}(G)\mathbf{u}| < \tau^2\lambda$ set the appropriate values to zero, while active nodes emit $\tau\mathbf{a}$ output spikes.

3.3.3 On-chip Memory

We now address recurrence on the TrueNorth chip in the context of the sub-function labeled "Compute τ^2u" calculated by Equation (3.4). Suppose we program this core by Figure 3.15 with inputs $\tau^2\mathbf{u}[n] = 2$ and $\tau^2\Delta\mathbf{u}[n] = 1$ resulting in outputs $\tau^2\mathbf{u}[n+1] = 3$ denoted by solid red and green circles respectively.

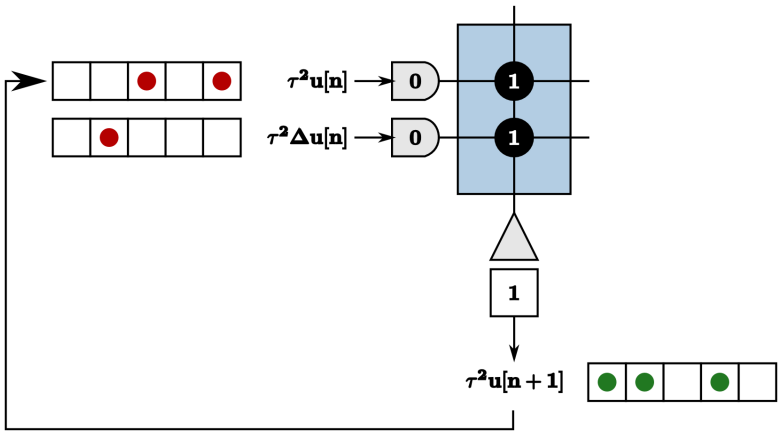

Figure 3.15: An example core to perform the node state update for one LCA node. Only the positive representations of inputs and outputs are shown.

As discussed in prior sections, we must wait a time window of length w to get true output values of neurons, but waiting is not a trivial task on the TrueNorth chip. We show where the output spikes $\tau^2\mathbf{u}[n+1]$ are sent in the appropriate time steps for the first axon in Figure 3.16. As a result, the output spikes begin to saturate our time window and incorrectly compute all values for future iterations.

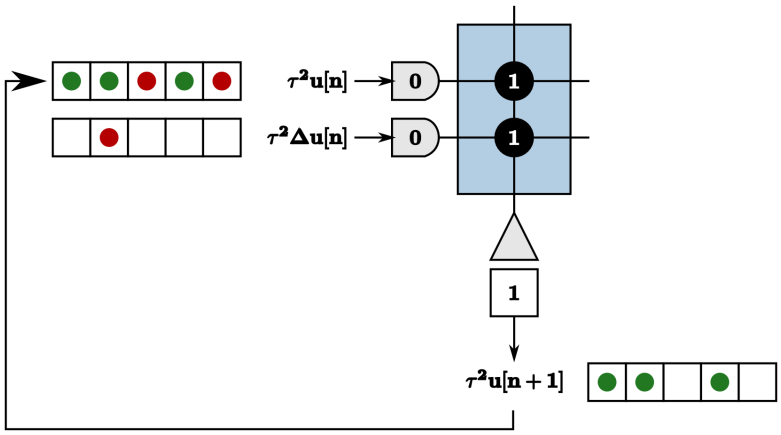

Figure 3.16: Recurrence resulting in incorrect computations of the node state update.

The TrueNorth chip does not have memory other than the constantly evolving neuron potentials. Therefore, we must leverage the many neuron parameters to emulate on-chip memory to store and retrieve high-precision values. We create a corelet using a group of

cores that computes values $\tau^2\mathbf{u}[n+1]$ on one path for a time window of w ticks while sending calculated values $\tau^2\mathbf{u}[n]$ from another in parallel.

For one LCA iteration consisting of w ticks, we allow spikes to be processed on one path while inhibiting any spikes to the other path. Suppose Path B is our inhibited path. We send the first core of Path B inhibitory spikes generated by on-chip triggers as input. Inhibitory spikes have a synaptic weight of negative one so that any incoming spikes from $\tau^2\Delta\mathbf{u}[\mathbf{n}]$ or $\tau^2\mathbf{u}[\mathbf{n}]$ are ignored since $1-1=0$ and a threshold of one is not met. Path A does not receive inhibitory input spikes to its first core for this iteration. The incoming spikes are therefore processed and sent to the second core on the same path.

The second core of Path A computes $\tau^2\mathbf{u}[n+1]$ using incoming spikes for w ticks. The second core of Path B sends the prior iteration's calculation $\tau^2\mathbf{u}[n]$ forward to the route core. We show two iterations in Figure 3.17 to visually describe the process.

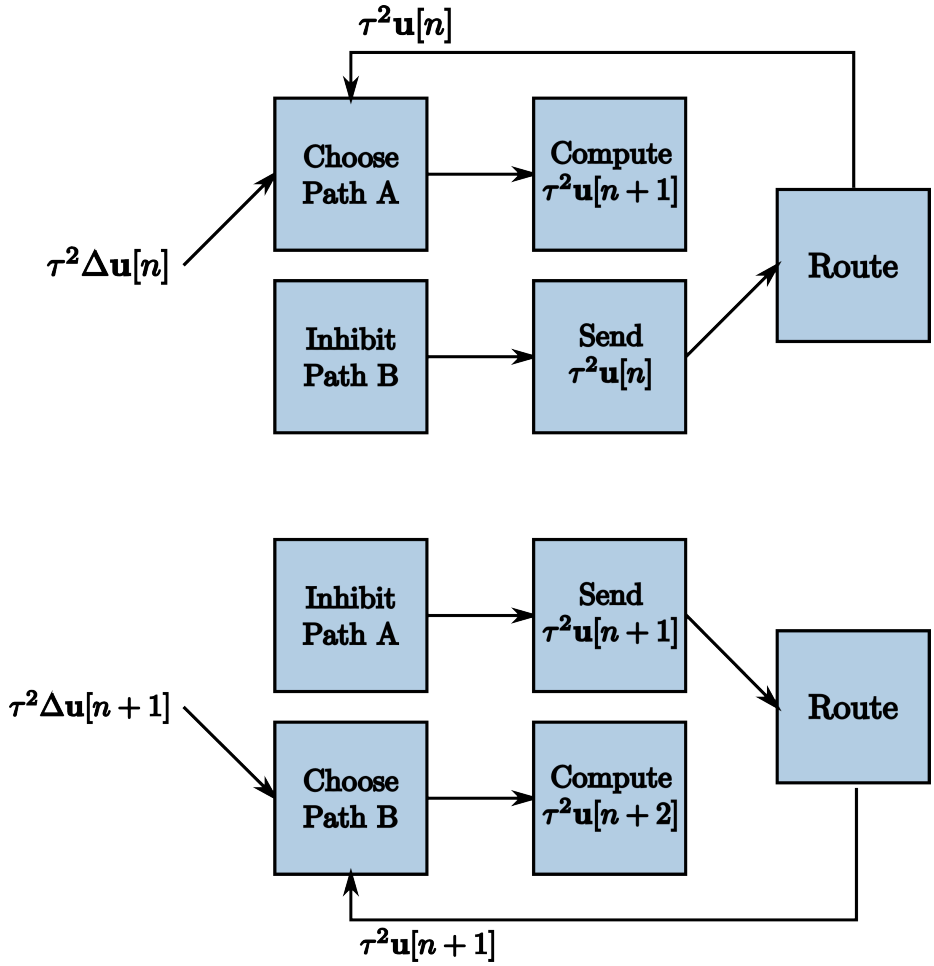

Figure 3.17: Two subsequent LCA iterations to accommodate recurrence in the system. Triggers enable one path to calculate $\tau^2\mathbf{u}[n+1]$ over w ticks while the other path sends the prior iteration's values $\tau^2\mathbf{u}[n]$ for use in the calculation.

We program this inhibition using synaptic weights of one for $\tau^2\Delta\mathbf{u}[\mathbf{n}]$ and $\tau^2\mathbf{u}[\mathbf{n}]$ and negative one for incoming inhibition spikes. An example core is shown in Figure 3.18. For the path that is not inhibited, there are no spikes in the axon labeled "Inhibit Trigger".

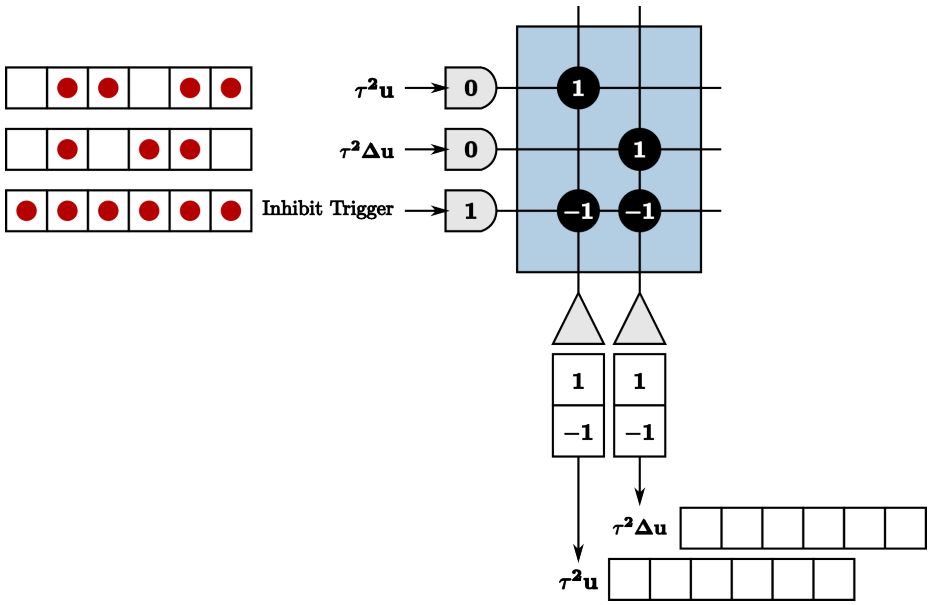

Figure 3.18: A programmed core to perform inhibition on the path that sends the prior iterations values.

We create a clock using TrueNorth neurons to generate triggers at the appropriate ticks within each iteration's windows. To inhibit spikes for w ticks, we need w spikes on one path to occur starting at even multiples of w and w spikes to occur starting at odd multiples of w for the other path. The largest synaptic weight we can choose is +255 and our window sizes are most times larger. We choose a leak of one and a positive threshold of $\alpha = 255$ for a first neuron and send the output spikes to a second neuron with a synaptic weight of one and a positive threshold $\alpha = w/255$. The output of the second neuron is therefore one spike every w ticks.

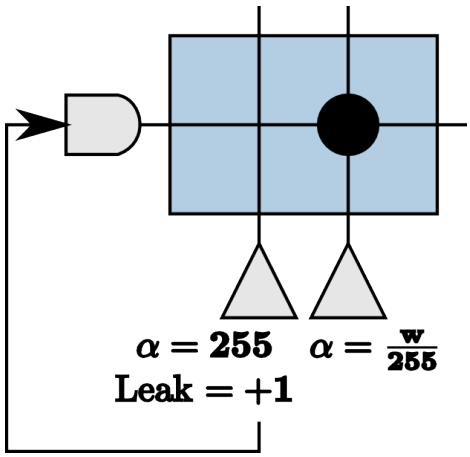

Figure 3.19: A programmed on-chip clock that emits a spike from the second neuron every w ticks.

To modify this clock to emit spikes on odd or even iterations only, we create two clocks and set the initial potentials accordingly, shown in Figure 3.20.

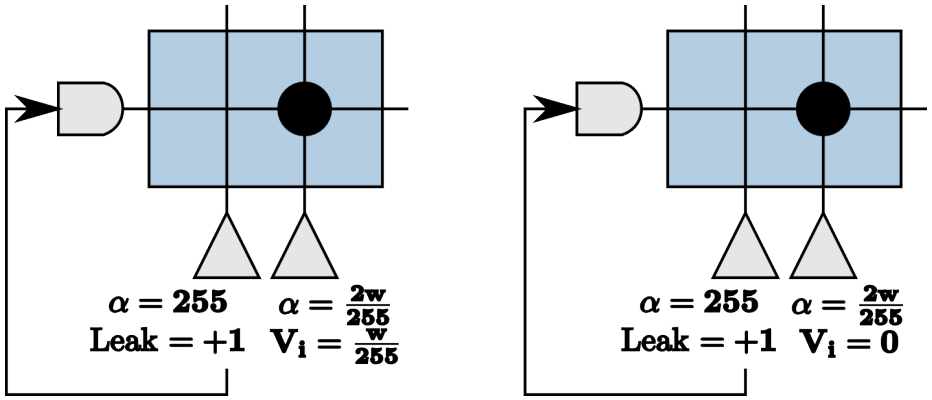

Figure 3.20: Programmed on-chip clocks. The clock on the left emits a spike every odd multiple of w and the clock on the right emits a spike every even multiple of w.

We use these clocks to generate the on-chip inhibition triggers by sending the output to neurons that constantly emit spikes if turned on and otherwise do not spike, shown in Figure 3.21. We use initial potentials of negative one and positive thresholds of $\alpha = 0$ for the inhibition trigger neurons. The inhibition trigger neurons are therefore turned on with an input spike given a synaptic weight of one and off with an input spike with a synaptic weight of negative one. The initialize neuron is used to begin the alternating inhibition triggers at the appropriate time step for inhibition to occur properly.

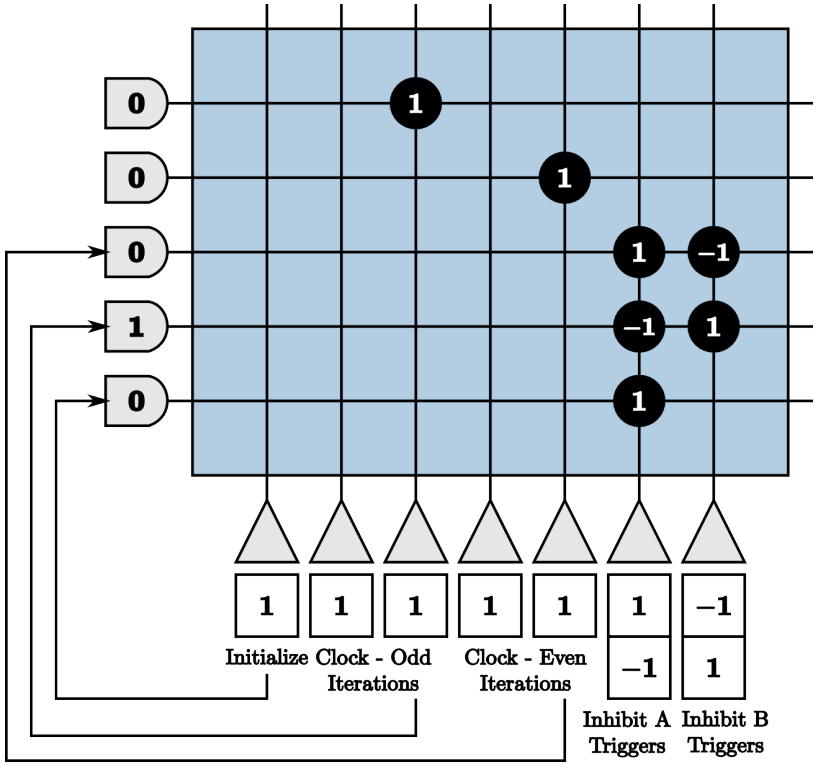

Figure 3.21: Programmed core that produces inhibition triggers.

The process of computing and sending information from the second core of each path requires a more complex set of on-chip triggers. To compute $\tau^2\mathbf{u}[n+1]$ in Path A's second core, we want our membrane potential to start at $-w$. All incoming spikes add and subtract for all neurons without reaching threshold until the time window is complete. After w ticks, the membrane potential of neuron m is $-w + \tau^2 u_m[n+1]$. To send spikes from Path B's second core in parallel, we send a trigger of user-generated spikes that increases the membrane potential by w. In the previous iteration, we compute $\tau^2 u_m[n]$ in Path B and store the value in the neuron potentials of the second core. With a linear reset and a threshold of one, Path B now sends $-w + \tau^2 u_m[n] + w = \tau^2 u_m[n]$ spikes to the route core. The route core sends these spikes back to Path A's first core to compute $\tau^2 u_m[n+1]$ in parallel. A depiction of the described neuron potential per on-chip triggers is depicted in Figure 3.22.

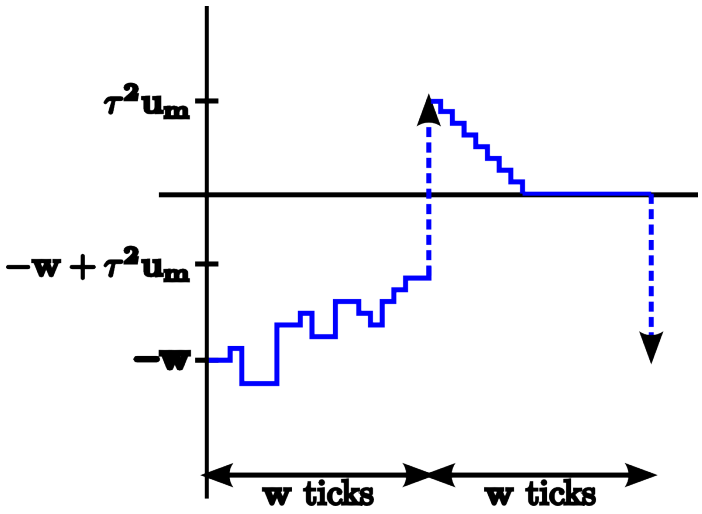

Figure 3.22: Neuron potential of one neuron for two iterations. The neuron computes $\tau^2 u_m[n+1]$ in the first window. During the second window, the potential is above zero and spikes emit from the neuron.

Once Path B's second core has sent all $\tau^2 u_m[n+1]$ spikes to the route core, the membrane potential settles to zero until the next iteration. However, we again have both positive and negative representations of the output of the sub-function, therefore our neuron potentials look like those shown in Figure 3.23 for the representation that does not spike given the threshold of one. Suppose Figure 3.22 is the positive representation of a neuron and appropriately calculates $\tau^2 u_m[n+1]$ as shown in the figure. The negative representation of the neuron is left with residual potential of $-\tau^2 u_m[n+1]$. When we move back to compute mode for the next iteration, we have a starting potential of $-w - \tau^2 u_m[n+1]$. Switching back to send mode, we see $-w - \tau^2 u_m[n+1] + \tau^2 u_m[n+2] + w = -\tau^2 u_m[n+1] + \tau^2 u_m[n+2]$ rather than the desired $\tau^2 u_m[n+2]$. If a change of sign occurs in a node's dynamics, this problem causes the system to converge to an incorrect solution.

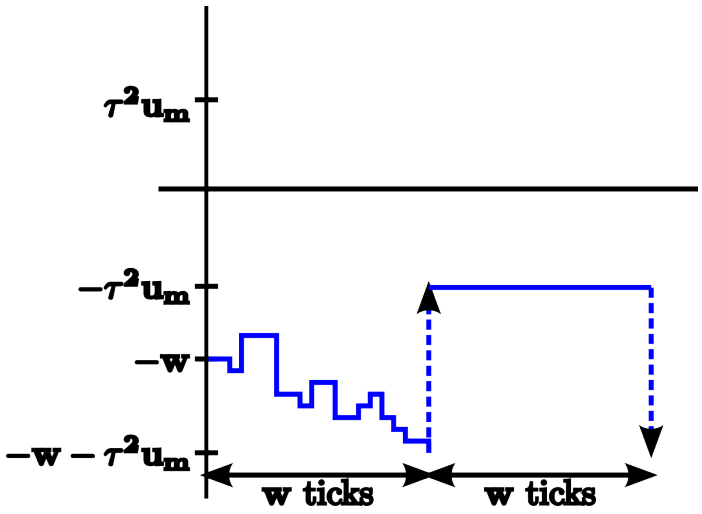

Figure 3.23: Neuron potential of one neuron for two iterations. This neuron computes the negative representation of $\tau^2 u_m[n+1] > 0$ in the first window. During the second window, the potential remains below zero. When the mode switches back to compute, computation is incorrect due to residual potential.

To correct the problem, we feedback any output spikes of the positive representation neurons to the negative representation neurons and vice versa with a synaptic weight of one as similar to what we described in Section 3.3.1. During the iteration where the core is sending information, the negative representation neuron potentials from our example will change to Figure 3.24, mirroring the positive representation neuron potentials. This starts the next count iteration correctly at $V_m = 0$ for both the positive and negative representations.

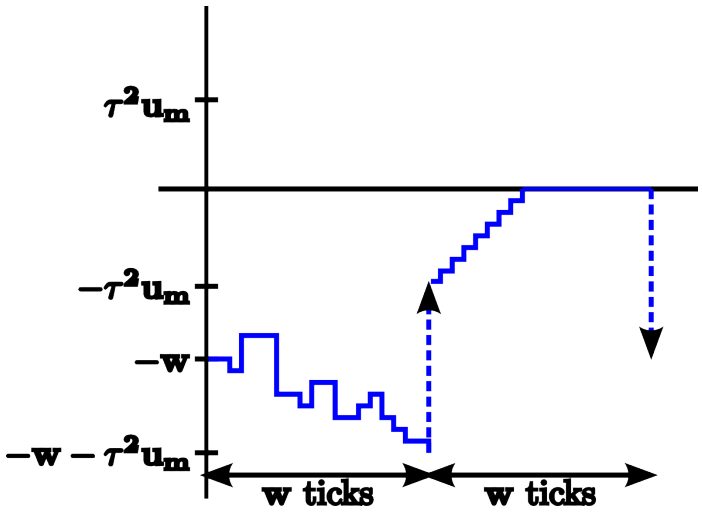

Figure 3.24: Neuron potential of one neuron for two iterations. This neuron computes the negative representation of $\tau^2 u_m[n+1] > 0$ in the first window. By feeding back spikes to correct for residual potentials, the potential increases to and remains at zero for accurate computation in the next window.

Because our largest positive and negative synaptic weights are integers $+255$ and -255 respectively, we set our window size as a multiple of 255 ticks for the entire system. We then send the required number of input spikes to send neuron membrane potentials to $-w$ and $+w$ for the compute and send modes by the triggers generated in Figure 3.25. Clock neurons are programmed as before in Figure 3.20. The different modes neurons represent triggers that activate one path while inhibiting the other and vice versa. The initialize neuron is again used to begin the triggers at the appropriate time step for spikes to align properly.

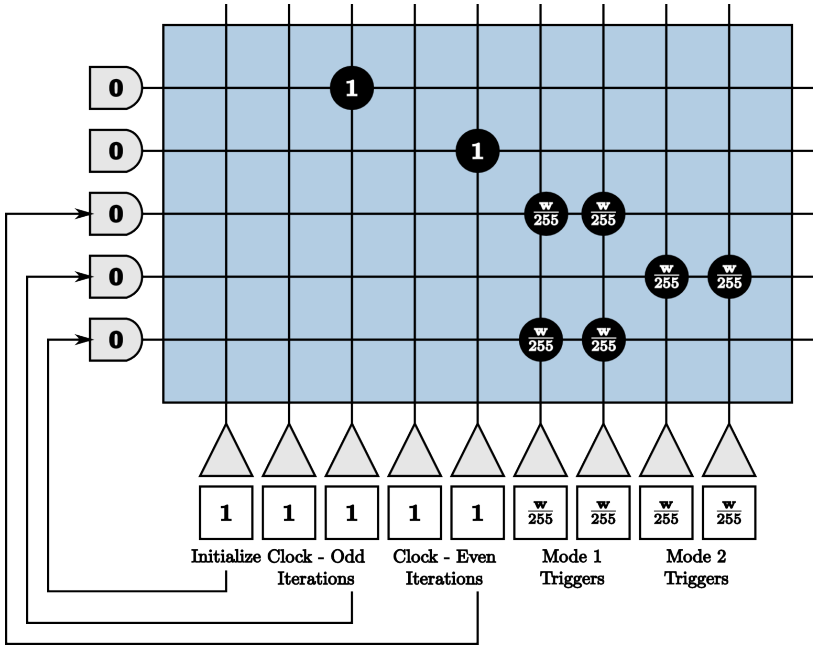

Figure 3.25: Programmed neuron that sends inhibition triggers.

Because these cores work together to perform a specific function, we deem this processing unit our on-chip memory corelet, denoted by the following diagram:

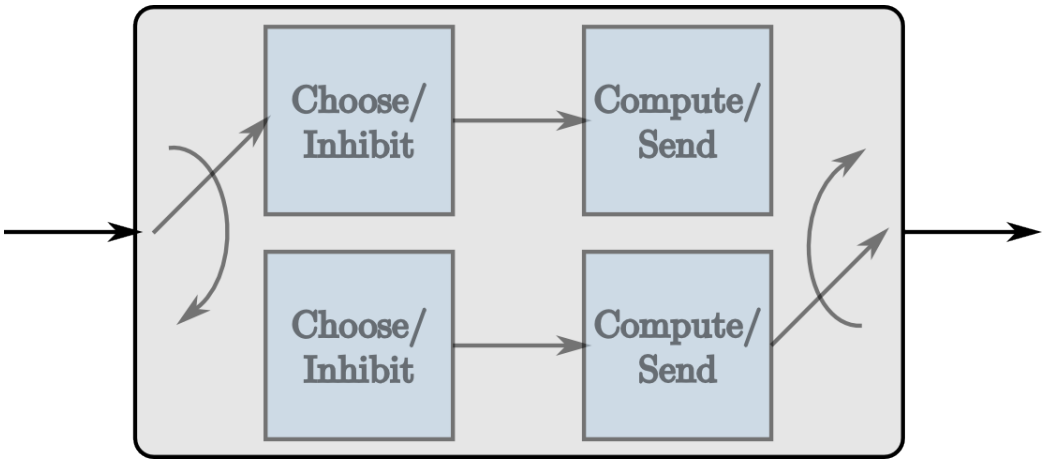

Figure 3.26: The on-chip memory corelet.

3.3.4 Vector-Matrix Multiplication with Increased Precision

Given there are only four synaptic weights per neuron and four axon types to choose from in a TrueNorth core, values of a matrix for a vector-matrix multiply (VMM) on TrueNorth

are restricted. For example, if we use synaptic weights to program a VMM on TrueNorth, we would transpose the matrix by which to multiply and assign synaptic weights the values of the matrix. However, any axons assigned the same type will take on the same synaptic weights if connected to the same neuron. We visualize the VMM

$$M \times v = \begin{bmatrix} 8 & -1 & 2 & 4 & 4 \\ 6 & 2 & -4 & 7 & 7 \\ -3 & 5 & 8 & -9 & -9 \\ 2 & -8 & 2 & -5 & -5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 3 \\ 1 \\ 2 \end{bmatrix}$$

in Figure 3.27. The last two axons are assigned the same type, therefore the last two columns in the multiplication matrix are identical.

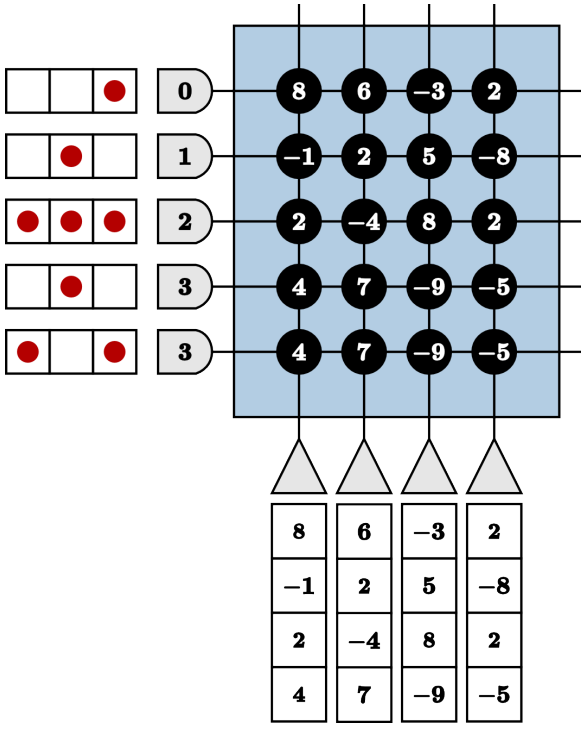

Figure 3.27: Restricted precision for vector-matrix multiplication on TrueNorth.

Consequently, we use three layers of cores in series to achieve 9-bits signed precision

multiplication matrices. The first layer essentially converts the integer values of the matrix to unsigned binary values. The second layer applies the first set of weights [8 4 2 1], to the first and second halves of the binary values. The third layer applies the significance of the most and least significant bits.

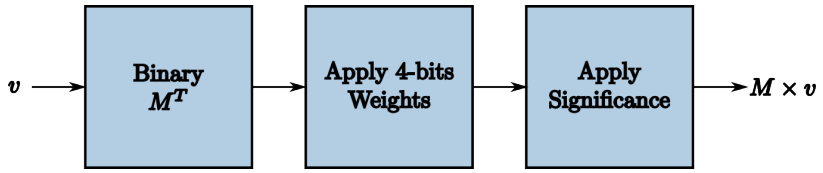

Figure 3.28: Layers required to increase vector-matrix multiplication to 9-bits signed precision on TrueNorth.

If we want to represent the value 146 using this method, the binary representation would be [1 0 0 1 0 0 1 0]. To program the TrueNorth chip with the binary representation, a binary zero signifies no synaptic connection, while a binary one denotes a connection between the axon and neuron in the core. In this layer, synaptic weights are one and thresholds are $\alpha = 1$ with linear resets to neuron potentials.

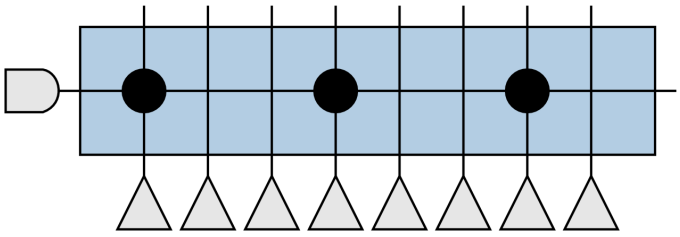

Figure 3.29: The first layer of our vector-matrix multiply representing a binary value of 146. Only the positive representations of inputs and outputs are shown.

The output of the first layer is sent to the axons of the second layer where weights are applied using programmed synaptic weights, shown in Figure 3.30. The thresholds are programmed as $\alpha = 1$ with linear resets to neuron potentials.

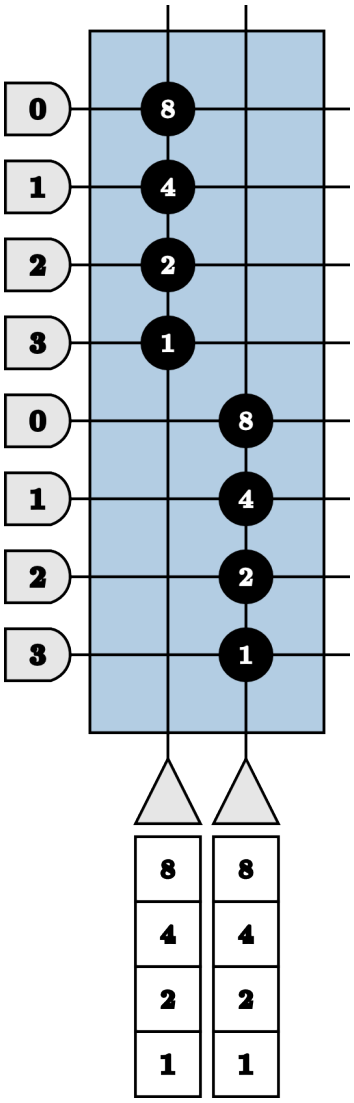

Figure 3.30: The second layer of our vector-matrix multiply where the first set of weights are applied to the binary values.

The output of the second layer is sent to the axons of the third layer where weights [16 1] are applied using programmed synaptic weights, shown in Figure 3.31. The thresholds are set again to $\alpha = 1$ with linear resets to neuron potentials.

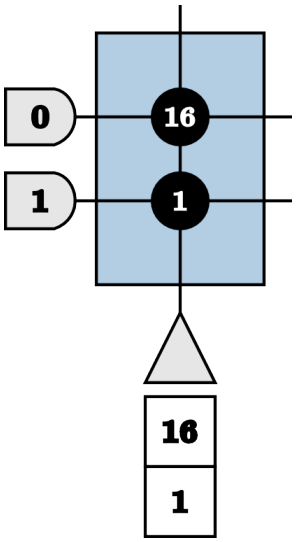

Figure 3.31: The third layer of our vector-matrix multiply where the final set of weights are applied to the binary values.

Suppose we send one spike as input to the axon in the first layer to multiply 146 by 1. Spikes emit from the first, fourth, and seventh neurons of the first layer and are sent to the first, fourth, and seventh axons of the second layer. The neurons in theh second layer emit nine and two output spikes, respectively. These spikes are routed to the third layer, where the final output accurately computes $9 \times 16 + 2 \times 1 = 146$.

To achieve this increased precision, a large number of neurons per value in the multiplication matrix are required. Suppose we would like to multiply a vector by the following matrix using our three-layer method.

$$\left[\begin{array}{rrrrrrrrr}
161 & 238 & 149 & -55 & 100 & -5 & -195 & 102 & 161 \\
207 & -175 & 235 & 79 & -93 & -28 & -1 & 200 & -131 \\
-191 & 240 & 80 & -168 & 230 & 75 & 235 & 235 & 219 \\
211 & 234 & -237 & 105 & -238 & 107 & -82 & 24 & -77 \\
68 & -7 & 178 & -239 & -31 & 130 & 44 & -185 & -155 \\
-206 & 153 & 222 & -114 & -61 & -114 & -141 & -179 & -127 \\
-113 & -183 & 91 & -232 & 136 & 92 & 128 & -124 & 59 \\
24 & -40 & 132 & -206 & 151 & 79 & -125 & 174 & -14 \\
234 & 212 & 124 & 165 & -160 & -172 & 3 & -126 & -76
\end{array}\right]$$

Eight neurons are needed per value for positive representation neurons, eight per negative representation neurons, and each of those must also be repeated to create symmetric thresholds as described in Section 3.3.1. For a 9×9 matrix, 288 neurons are needed and cannot be accommodated using the available 256 neurons in one core. For even larger matrices, the number of required axons exceeds the 256 available axons within one core. We therefore split our matrix into sub-matrices for each layer and implement each sub-matrix in separate cores in parallel. While we can accommodate this matrix with fewer cores that we show, we split our example matrix into nine matrices to clearly visualize the process we use to accommodate large matrices in Figure 3.32.

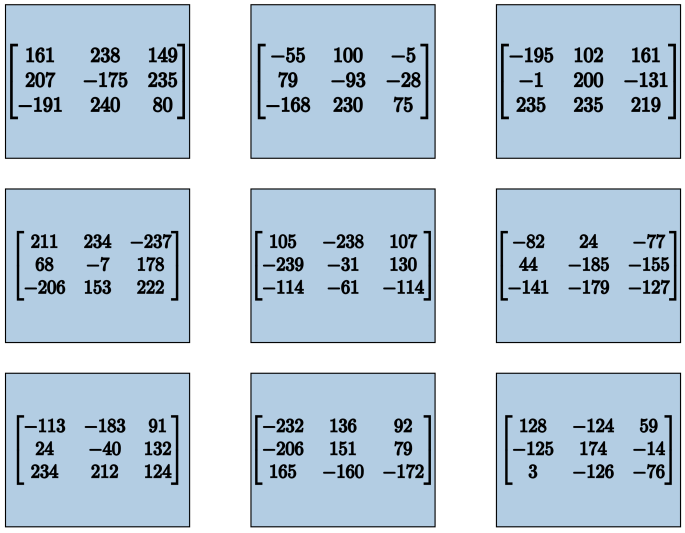

Figure 3.32: A matrix too large to accommodate with one core using our vector-matrix multiply method. This matrix is split to be programmed on multiple cores that operate in parallel.

The first and second layers are implemented as before for each sub-matrix independently. We denote this in Figure 3.33. Inputs to each of these sub-matrices' first layers must be repeated using a splitter as each neuron can only have one destination axon. A simple splitter is shown in Figure 3.34, where an input is repeated as many times as necessary using repeated neurons with destination axons assigned accordingly.

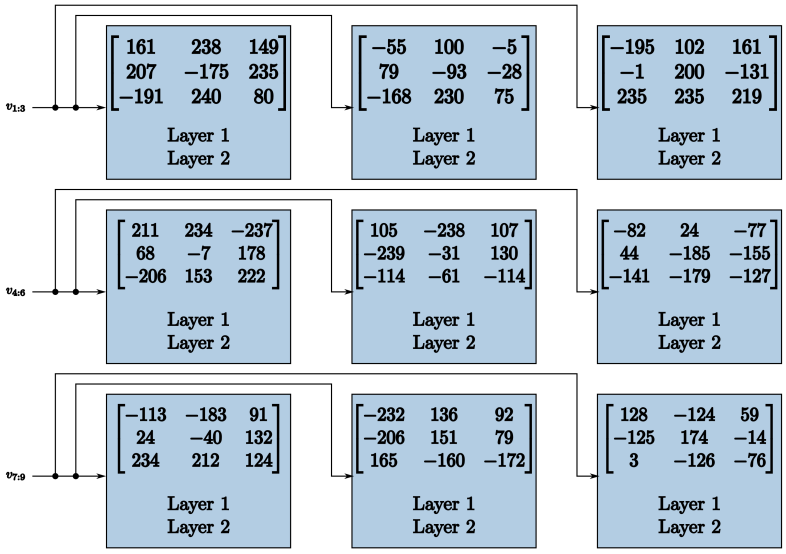

Figure 3.33: Inputs are repeated and sent to the appropriate sub-matrices. The first and second layers are programmed as before for each sub-matrix.

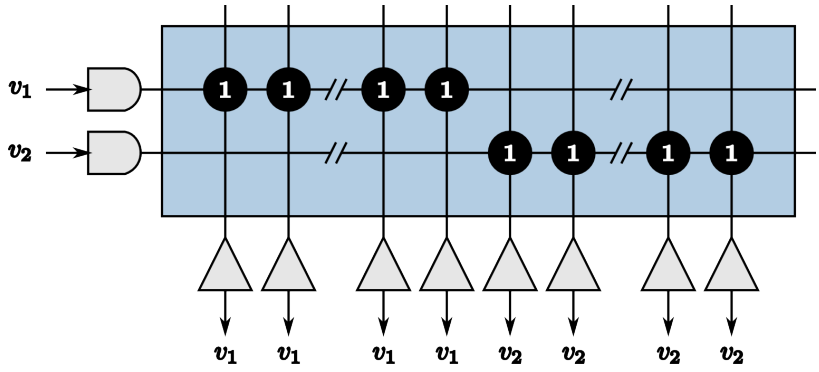

Figure 3.34: Inputs are repeated and sent to the appropriate sub-matrices. The first and second layers are programmed as before for each sub-matrix.

The third layer is used to not only apply the significance of the binary values in the matrix, but also to reunite each column of the vector-matrix multiply. The third layer produces positive and negative representations of resultant values for each column of the VMM in parallel. The output spikes from each layer three core are concatenated for the full vector solution.

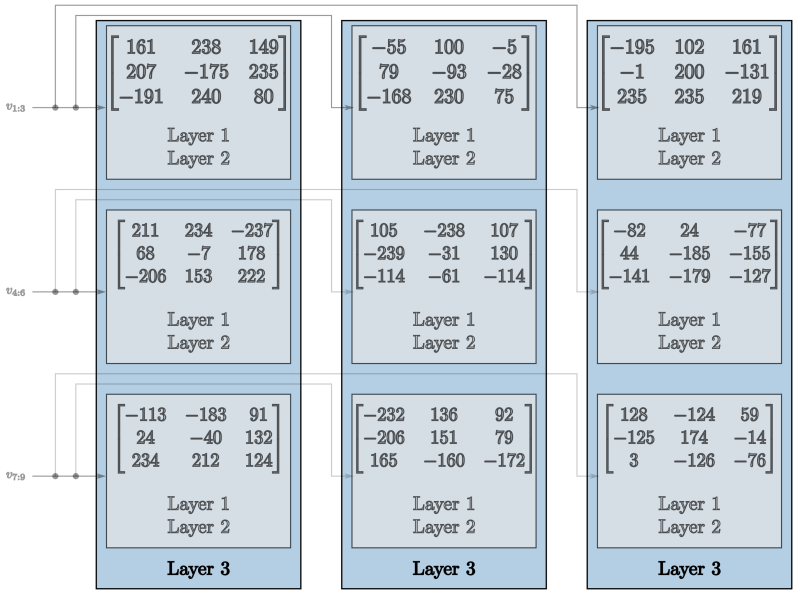

Figure 3.35: Inputs are repeated and sent to the appropriate sub-matrices. The first and second layers are programmed as before for each sub-matrix.

Because these cores work together to perform a specific function, we deem this processing unit our 9-bits signed VMM corelet, denoted by the following diagram:

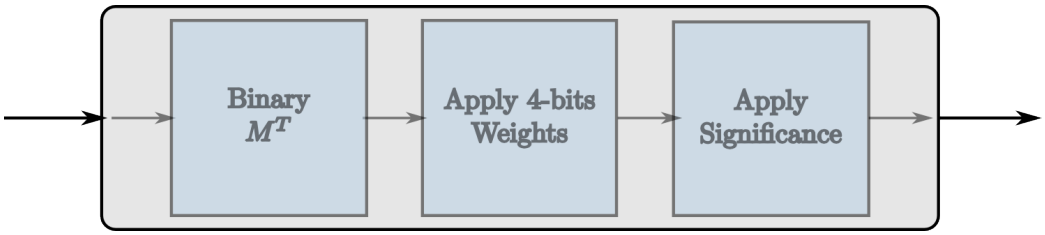

Figure 3.36: The 9-bits signed VMM corelet.

CHAPTER 4

THE LOCALLY COMPETITIVE ALGORITHM IMPLEMENTED ON THE TRUENORTH CHIP

We combine the processing units described in Chapter 3 to implement the Locally Competitive Algorithm (LCA) system on the TrueNorth chip. Our results provide the largest LCA system on neuromorphic hardware to date. We show that by using our novel design methodology, that the LCA can be exactly implemented on the TrueNorth chip and therefore convergence to the correct sparse approximation is guaranteed.

At our disposal, we have a non-linear soft threshold, an on-chip memory corelet, and a 9-bits signed vector-matrix multiply corelet, each described in detail in Chapter 3. The LCA is scaled by τ^2 so that values can be accurately computed using the integer precision on the TrueNorth chip. A block diagram of the scaled LCA can be seen in Figure 4.1.

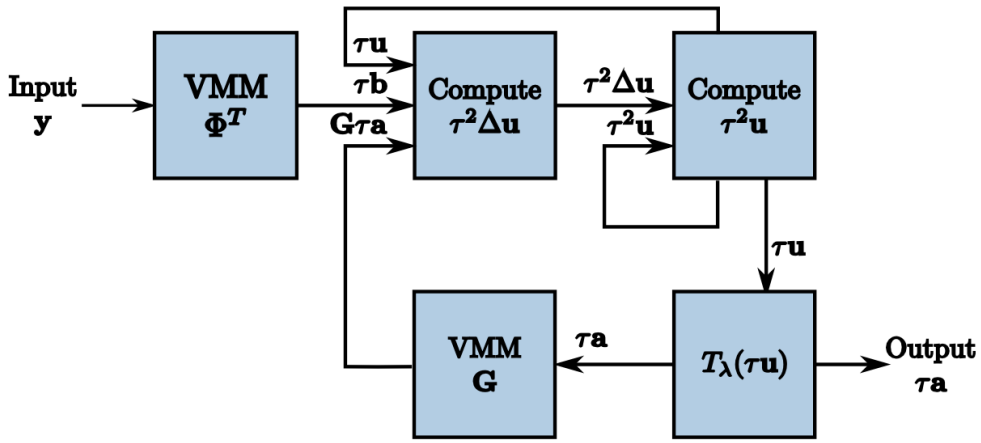

Figure 4.1: The scaled LCA broken into sub-functions.

The input $\mathbf{y}$ to the system is always the same. The initial projection b is therefore

also the same, and must be repeated at the beginning of every time window for accurate computation of each LCA iteration. We use the principles from creating on-chip memory in Section 3.3.3 to repeat the initial projection values periodically on the hardware. The user therefore only sends the signal as input for one iteration, our 9-bits signed vector-matrix multiply corelet computes the initial projection once, and we implement a periodic repeater to send the values at the beginning of every time window, shown in Figure 4.2.

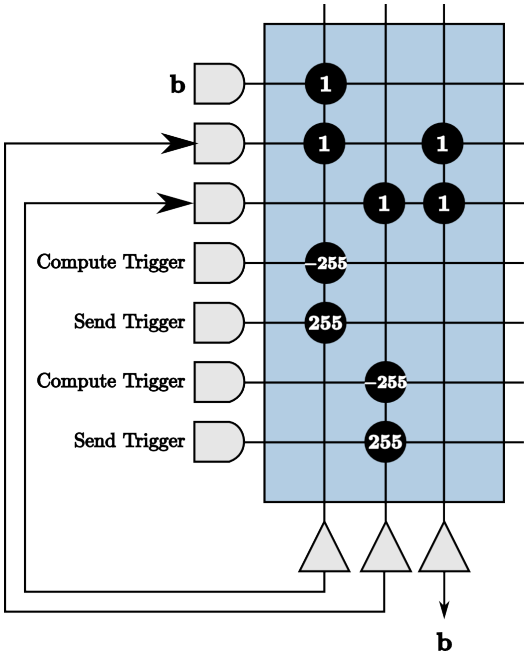

Figure 4.2: The initial projection repeated on-chip using principles from our on-chip memory corelet.

Similarly, we offer the soft threshold $\tau^2\lambda$ as a user input to our system such that the trade-off between reconstruction error and sparsity of the sparse approximation of a signal can be determined at run-time by the user. We implement this periodic repetition in the same way as we do for the initial projection.

Our LCA corelet therefore becomes the following:

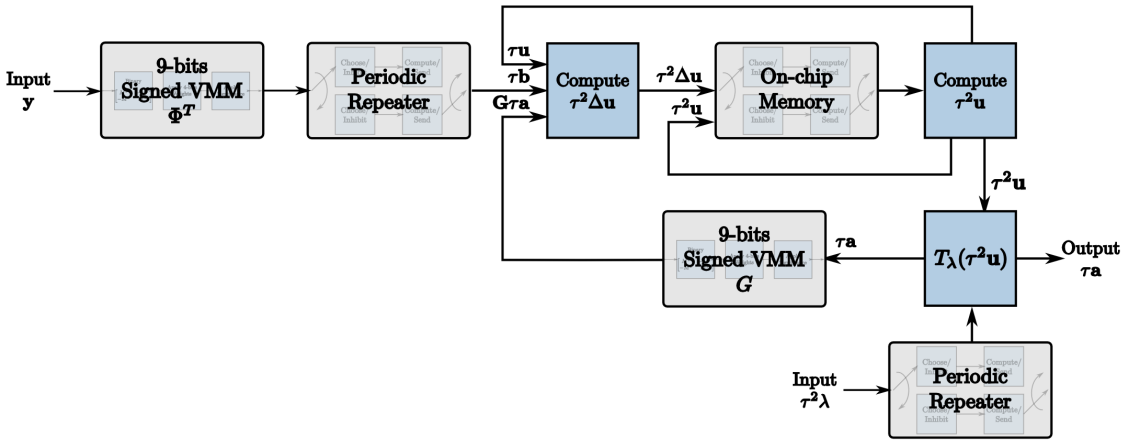

Figure 4.3: The corelet used to implement the Locally Competitive Algorithm on the TrueNorth chip using our novel design methodology.

4.1 Performance Results

We demonstrate the success of the LCA on TrueNorth for dictionaries containing randomly distributed values {-1,0,1} with up to 100 nodes, valuable for use in signal processing applications such as compressed sensing that use image patches as input data. We compare the LCA node dynamics $\mathbf{u}$ computed by the TrueNorth chip to the dynamics of a discrete LCA system. We consider our results a success if the LCA node dynamics match, proving that our LCA system on the TrueNorth hardware converges to the same and therefore correct sparse approximation of the original signal.

To compare and evaluate the node dynamics of each LCA system, we repeat neurons with thresholds of one within the sub-function labeled "Compute τ^2u" and send those output spikes as outputs of the system. We count the number of spikes in each window of w ticks to determine the evolving values at each iteration. The output of the TrueNorth chip for an example of a 50-node system is shown in Figure 4.4. Pins 1 to 50 and pins 51 to 100 are outputs that represent the positive and negative representations of $\tau^2\mathbf{u}$ respectively. We observe that pins 16 and 86 are the only pins that emit spikes as the system evolves over time, while other pins spike early on but decrease to zero. This is a clear depiction of LCA

nodes competing and inhibiting one another to converge to the correct sparse approximation.

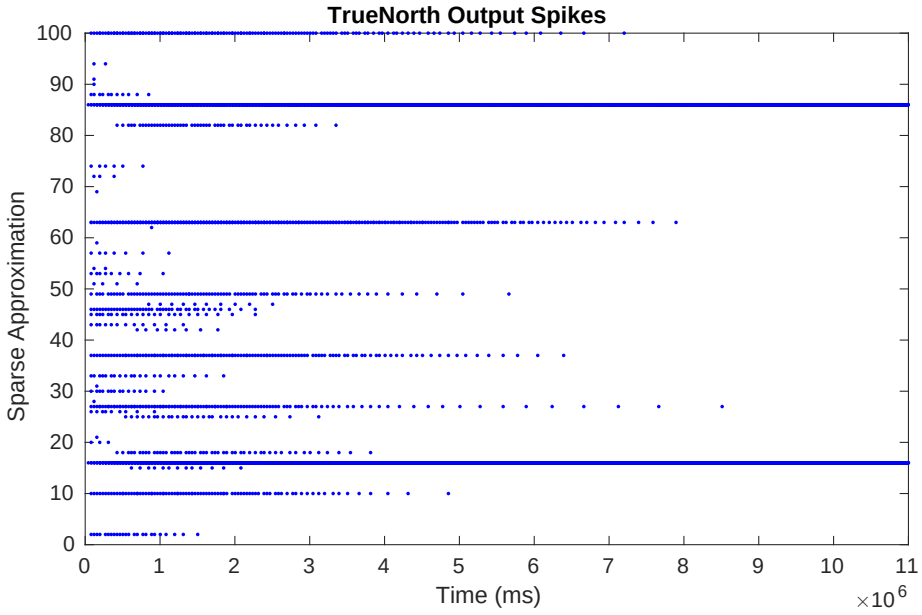

Figure 4.4: The output spikes from the LCA corelet on the TrueNorth chip, representing positive and negative representations of $\tau^2\mathbf{u}$.

The values calculated by counting spikes within each window are divided by $\tau^2\text{diag}(G)$ and overlaid onto a discrete LCA system computing the node dynamics with the same parameters. Node dynamics falling between the dashed lines denote those that fall below the LCA threshold and therefore do not contribute to signal reconstruction. Our system produces identical curves to that of the discrete LCA for hundreds of examples with randomly chosen dictionaries, random input signals generated by a linear combination of one to five dictionary nodes with random weights, and randomly chosen parameters τ and λ, proving that we correctly compute the sparse approximation of a signal for LCA systems. The node dynamics in Figure 4.5 are the results of the spikes from Figure 4.4.

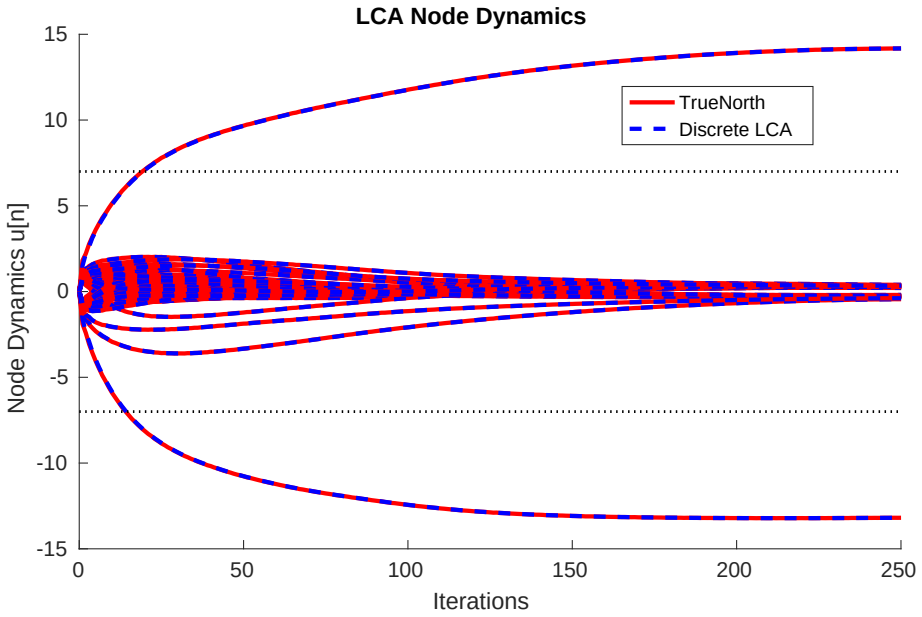

Figure 4.5: The node dynamics of an LCA system with a 33×50 dictionary compared to a discrete LCA system. Input signals are $y = 14 \times \Phi_{16} - 13 \times \Phi_{36}$ and parameters are $\tau = 13$ and $\lambda = 7$.

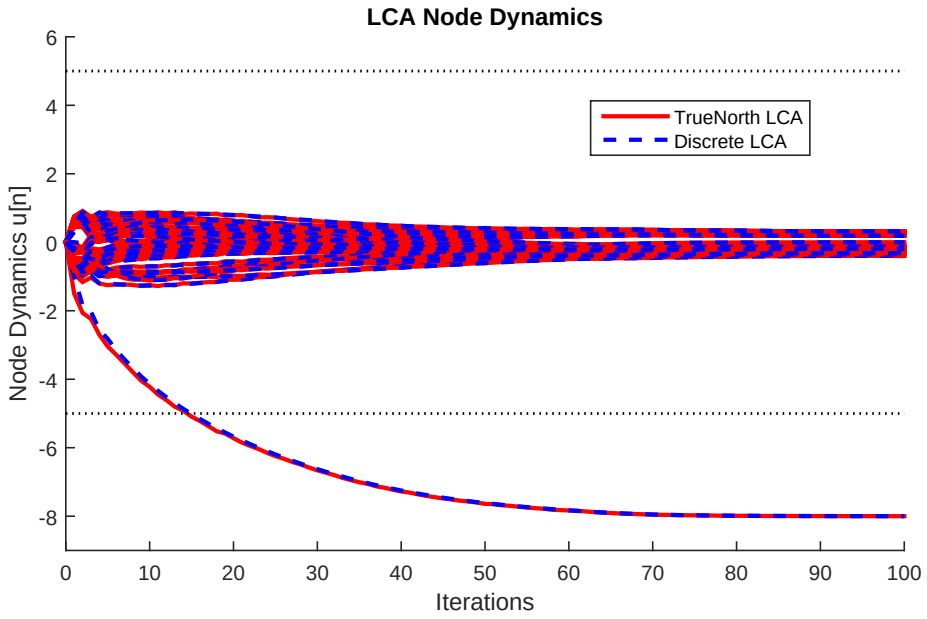

Figure 4.6: The node dynamics of an LCA system with a 20×41 dictionary compared to a discrete LCA system. Input signals are $y = -8 \times \Phi_{19}$ and parameters are $\tau = 8$ and $\lambda = 5$.

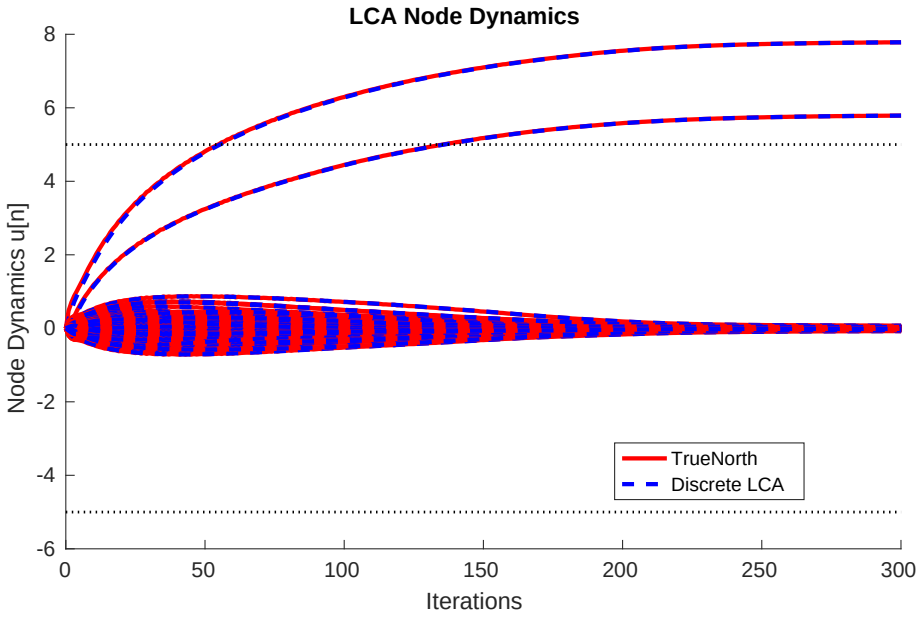

Figure 4.7: The node dynamics of an LCA system with a 66×100 dictionary compared to a discrete LCA system. Input signals are $y = 8 \times \Phi_{16} + 6 \times \Phi_{52}$ and parameters are $\tau = 18$ and $\lambda = 5$.

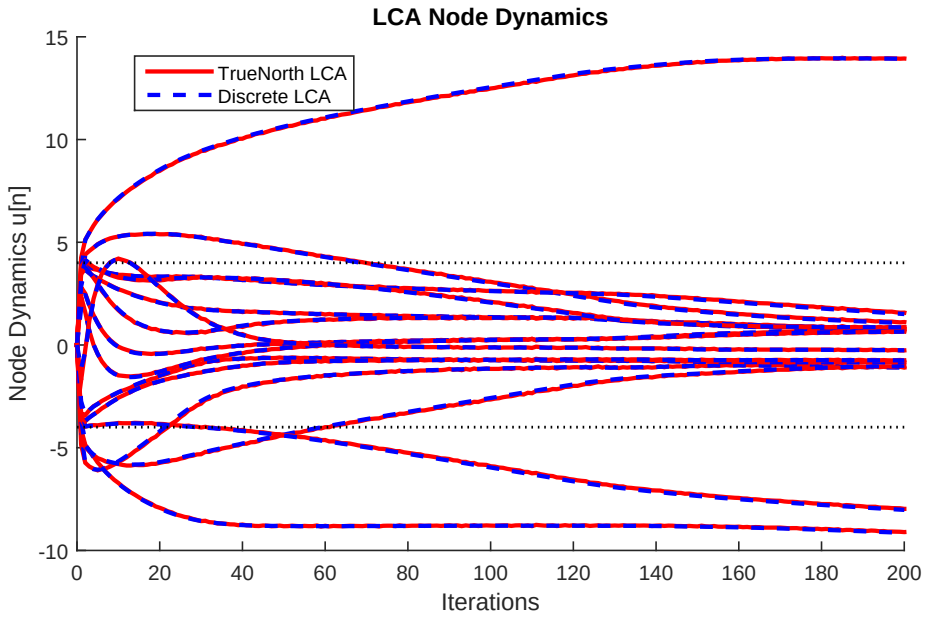

Figure 4.8: The node dynamics of an LCA system with a 6×15 dictionary compared to a discrete LCA system. Input signals are $y = 13 \times \Phi_3 - 9 \times \Phi_7 - 8 \times \Phi_{10}$ and parameters are $\tau = 18$ and $\lambda = 5$.

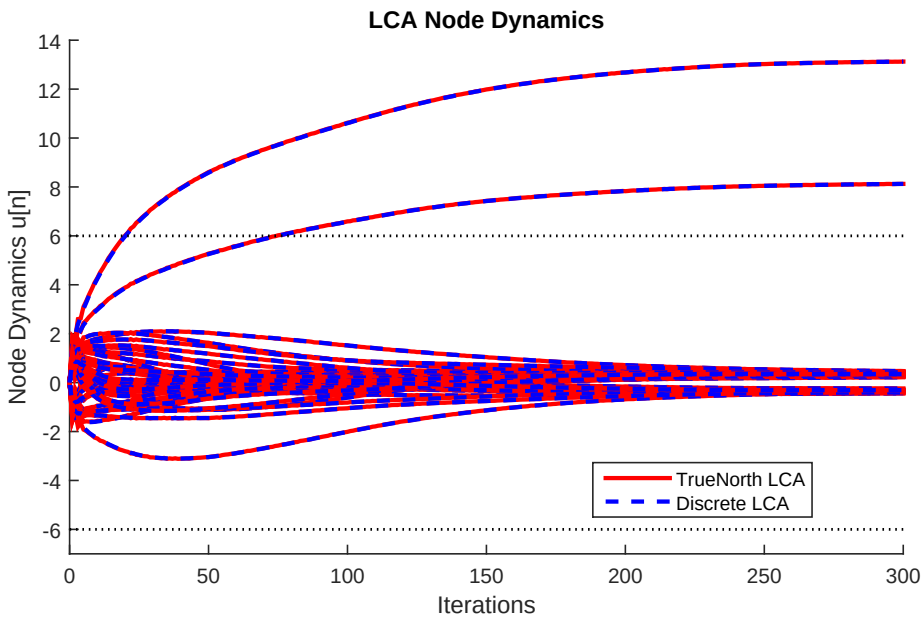

Figure 4.9: The node dynamics of an LCA system with a 22×45 dictionary compared to a discrete LCA system. Input signals are $y = 8 \times \Phi_{33} + 13 \times \Phi_{42}$ and parameters are $\tau = 11$ and $\lambda = 6$.

4.2 Chip Utilization

Each sub-function of the LCA has a limit on the number of LCA nodes it can support if using one core per sub-function. Each core is limited to the aforementioned 256 axons and 256 neurons. By careful indexing, we can expand each sub-function across multiple cores that operate in parallel. Each chip is limited to 4096 cores, however chips can be tiled for seamless expansion of our system to accommodate any size of an LCA system.

Throughout our corelet, each individual neuron emits spikes and each axon receives spikes representative of the positive or negative representation of an LCA variable for a specific LCA node. However, connections between axons and neurons might not be constrained to the same LCA node. For example, in a vector matrix multiply, we see that a single input is connected to neurons that are representative of several distinct node's LCA variables.

We call axons that connect to neurons that represent the same LCA node as itself *unique axons* and denote the number of such axons per LCA node as Ax_u. We call those axons

that connect to more than one LCA node *common axons* and denote the number of such axons as Ax_c. We denote the number of TrueNorth neurons required per LCA node within a sub-function as N. The maximum number of LCA nodes we can represent in a single core for a specific sub-function is as follows:

$$\text{Maximum LCA nodes} = \min\left(\frac{256 - \text{Ax}_c}{\text{Ax}_u}, \frac{256}{N}\right) \tag{4.1}$$

We show the limits for each sub-function's cores based on these parameters in Table 4.1.

Table 4.1: Limitations on the number of LCA nodes a sub-function can accommodate per core.

Sub-function	N	Ax_u	Ax_c	Maximum LCA nodes
VMM - Initial Projection, Layer 1	32	16	Length($\mathbf{a}$)$\times 2$	8
VMM - Initial Projection, Layer 2	16	8	0	16
VMM - Initial Projection, Layer 3	4	4	0	64
Update $\tau^2\Delta\mathbf{u}$	6	8	0	32
Update $\tau^2\mathbf{u}, \tau\mathbf{u}$, Paths' 1st Cores	4	4	1	63
Update $\tau^2\mathbf{u}, \tau\mathbf{u}$, Paths' 2nd Cores	4	6	2	42
Update $\tau^2\mathbf{u}, \tau\mathbf{u}$, Route Core	10	2	0	25
Soft Threshold	4	2	1	64
VMM - Inhibition, Layer 1	32	16	Length($\mathbf{a}$)$\times 2$	8
VMM - Inhibition, Layer 2	16	8	0	16
VMM - Inhibition, Layer 3	4	4	0	64

We maximize the use of these sub-function's cores by programming the maximum LCA nodes to each sub-function's core. This minimizes the resources required to run the LCA on the TrueNorth chip, leaving over 4000 cores available for other processing tasks if using a dictionary with 100 LCA nodes. We calculate the number of cores that would be required for much larger dictionaries and show where we begin to exceed operation on one chip by any dictionary sizes that lie to the right of the magenta line in Figure 4.10. For reference, we show a marker at a dictionary with 66 inputs and 100 LCA nodes requiring 113 cores, falling far below the available 4096 cores available.

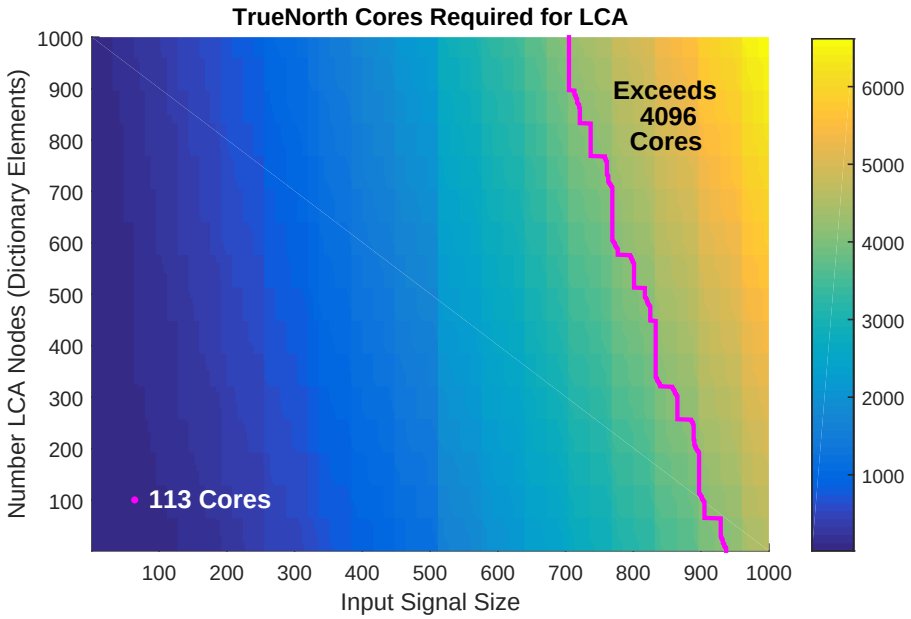

Figure 4.10: The number of cores required for LCA systems with dictionaries with up to 1000 inputs and outputs. Any dictionary sizes that lie to the right of the magenta line require more than one chip to be tiled for implementation.

4.3 Power Consumption

We measure power for several dictionary sizes and show results in Table 4.2. The total power is calculated by scaling the leakage power by the number of cores actually used, where P is power and $P_{total} = P_{active} + P_{leak} * N_{cores}/4096$ [15]. This low-power consumption offers our implementation of the LCA on TrueNorth as a feasible choice for embedded systems signal processing applications.

Table 4.2: Power consumption of the LCA implemented on the TrueNorth chip.

	Average Total Power (mW)	
Dictionary Size	Operating at .8V	Operating at 1V
12×18	.343	.726
33×50	.623	1.326
66×100	1.657	3.537

CHAPTER 5

ALTERNATE INFERENCE PROBLEMS ON THE TRUENORTH CHIP

The Locally Competitive Algorithm (LCA) architecture can be extended for use in other sparsity-based problems [28]. We offer a methodology to combine the processing units described in Chapter 3 for additional sparsity-based inference problems on the TrueNorth chip using these architectural extensions. Signal processing algorithms of interest feasible for implementation on the TrueNorth architecture are least squares, re-weighted ℓ_1, and approximate ℓ_0 minimization. Each of these problems can be solved using the LCA architecture by changing the regularization term. This involves modifying the non-linear threshold processing unit in our system.

5.1 Least Squares

The least squares method, commonly used in data-fitting applications, can be implemented using the LCA architecture by removing the soft threshold function entirely, resulting in $a_m = u_m$ for unit norm dictionary vectors. Given data points $\mathbf{x}$ and $\mathbf{y}$, the dictionary used in the LCA architecture for quadratic least squares is the following:

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{bmatrix}$$

and the resulting line of best fit is $\mathbf{y} = a_1 + a_2\mathbf{x} + a_3\mathbf{x^2}$. To program this problem on the TrueNorth chip, we program the cores using the dictionary above and remove the soft-threshold cores, shown in Figure 5.1.

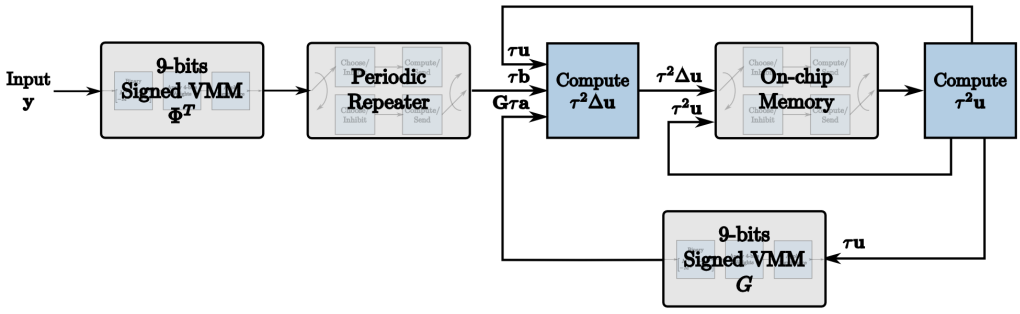

Figure 5.1: The corelet used to implement quadratic least squares using the Locally Competitive Algorithm architecture on the TrueNorth chip using our novel design methodology.

We implement this corelet on the TrueNorth chip and find the results in Figure 5.2. We observe error in the comparisons of the lines of best fit $\mathbf{y} = a_1 + a_2\mathbf{x} + a_3\mathbf{x^2}$. This is due to the integer precision available to represent data points. While one might expect we could scale this system as we scaled the LCA, the values in inhibition matrix $G = \Phi^T\Phi$ grow too large to be performed using our 9-bits signed VMM processing unit.

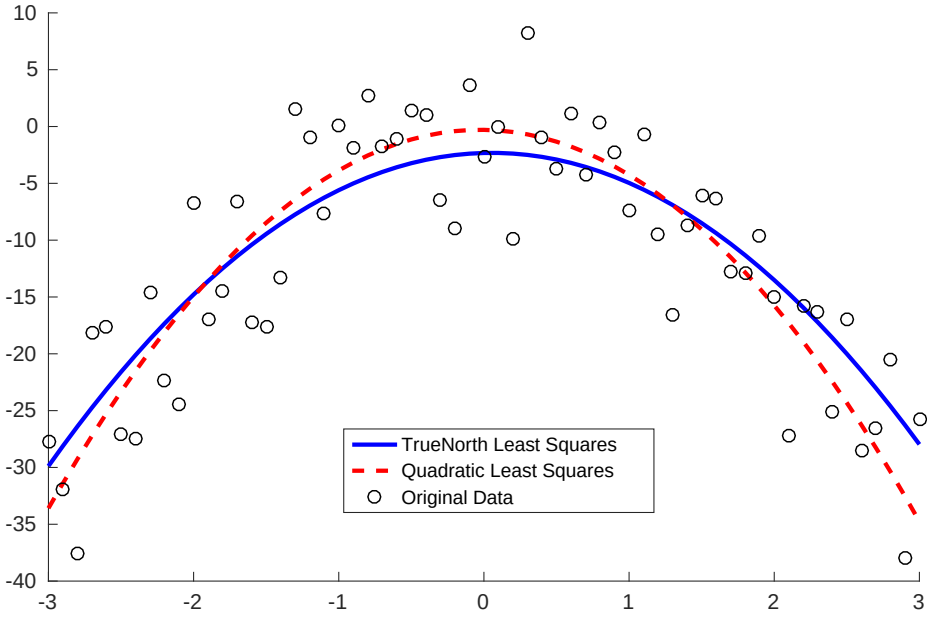

Figure 5.2: Quadratic least squares solved by the least squares corelet on the TrueNorth chip.

If we instead round the data points and perform quadratic least squares, we see zero error in the resulting line of best fit in Figure 5.3. This serves as a successful proof of concept quadratic least squares system on the TrueNorth chip. The precision of values for the multiplication matrix can be increased by using more neurons for each value in the matrix using the principles we provided in Section 3.3.4 if a priority is placed on exactly representing data points.

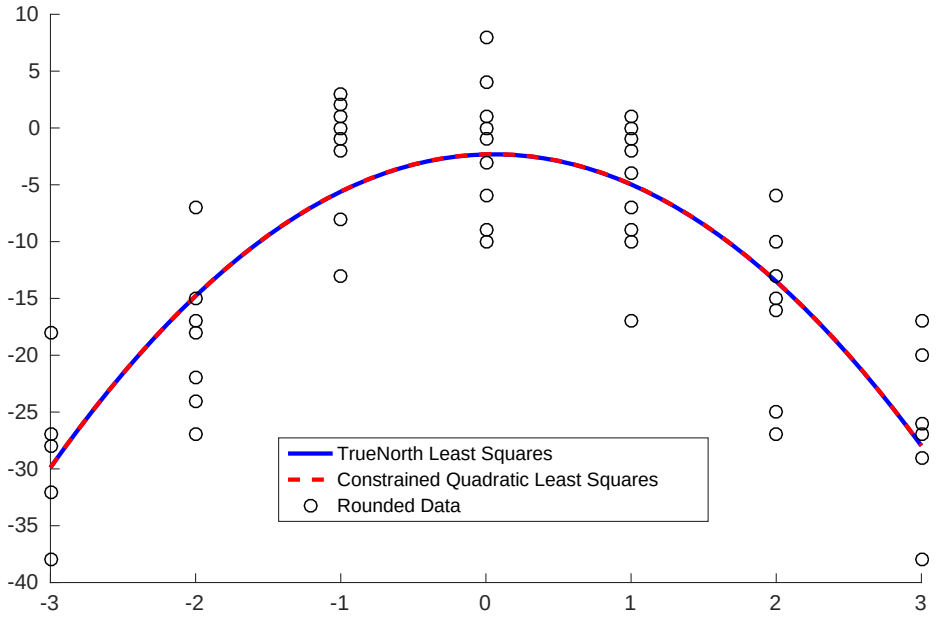

Figure 5.3: Quadratic least squares solved by the least squares corelet on the TrueNorth chip using data points rounded to the nearest integer values.

5.2 Re-weighted ℓ_1

The re-weighted ℓ_1 problem has proven useful in compressed sensing applications, in that with fewer observations of the original signal the resulting sparse approximation can decrease reconstruction error [29]. The re-weighted ℓ_1 problem is implemented using a set of dynamics placed on λ for each LCA iteration where ν is a proportionality constant, γ is a small parameter, τ_λ is a time constant specific to λ, and k denotes values for the k^{th} LCA

node [28].

$$\tau_\lambda \lambda_k[n+1] = \lambda_k^{-1}[n] - \nu^{-1}(|a_k[n]| + \gamma) \tag{5.1}$$

Because we generate inputs for each LCA iteration to send to the soft threshold off-chip, $\tau^2\lambda$ can be dynamically updated rather than a constant value from iteration to iteration to solve this problem. We show how one would utilize our processing units to implement solve the re-weighted ℓ_1 problem on the TrueNorth chip in Figure 5.4. We highlight the modified non-linear threshold using orange squares.

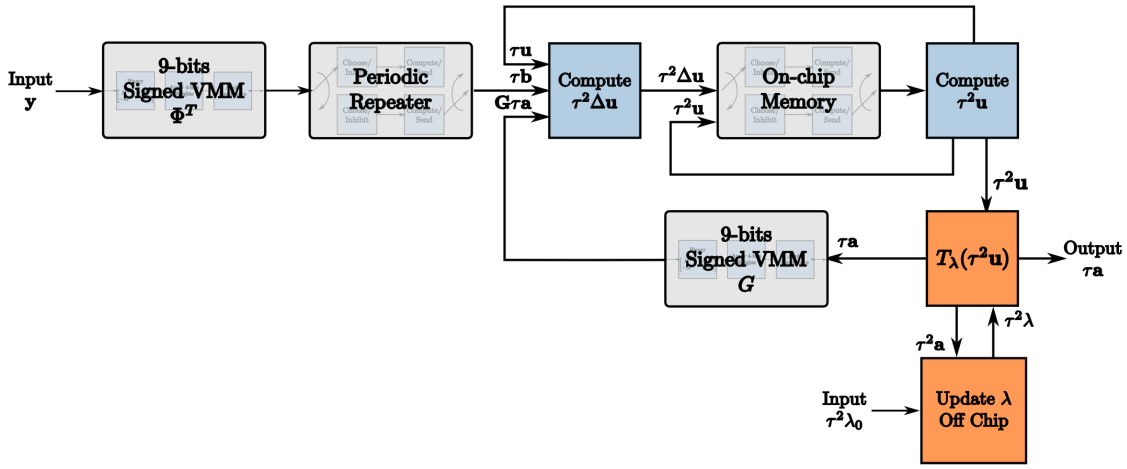

Figure 5.4: The corelet used to solve the re-weighted ℓ_1 problem using the Locally Competitive Algorithm architecture on the TrueNorth chip using our novel design methodology.

5.3 Approximate ℓ_0 Minimization

To implement approximate ℓ_0 minimization using the LCA architecture, the soft threshold function is replaced by the hard-threshold function [4] in Equation (5.2).

$$a_m(t) = T_\lambda(u_m(t)) = \begin{cases} \frac{u_m(t)}{g_m} & \text{if } |u_m(t)| \geq \lambda \\ 0 & \text{if } |u_m(t)| < \lambda \end{cases} \tag{5.2}$$

This approach yields weights that produce a perfectly reconstructed signal for an over-

complete dictionary. These weights however may not be the sparsest solution, but are similar to results of the matching pursuit algorithm which still produce reasonably sparse solutions [4].

To program the TrueNorth chip using a hard-threshold function, we use the same non-linear threshold processing unit and combine it with a periodic repeater generated using the principles for on-chip memory in Section 3.3.3. Using our non-linear threshold, we compute $\tau^2 u_m[n] - \text{sign}(u_m[n])\tau^2\lambda$ for values that exceed the soft threshold. To adjust for the hard threshold, we must add back $\text{sign}(u_m[n])\tau^2\lambda$ that was subtracted. We store this value in our periodic repeater and only send it if an activation neuron lets the repeater know that the soft threshold has been reached. The corelet depicting this process is shown in Figure 5.5.

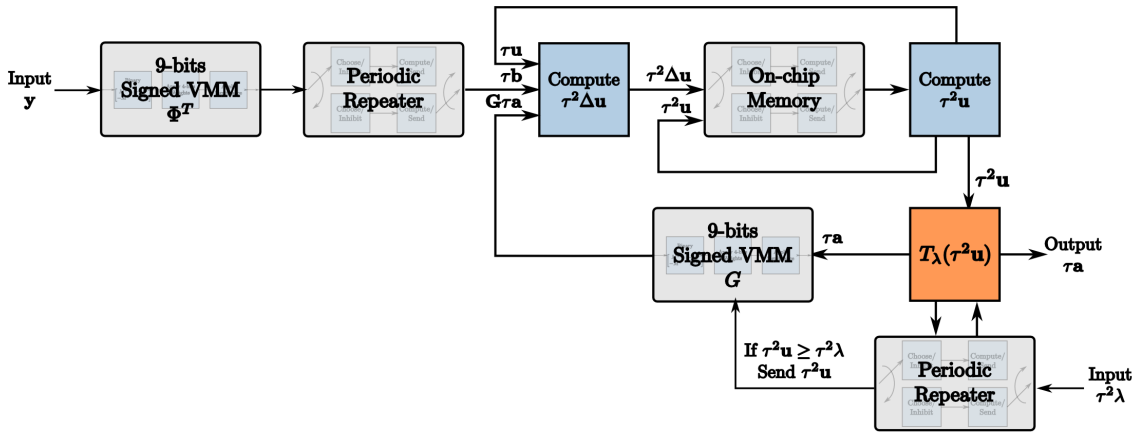

Figure 5.5: The corelet used to implement approximate ℓ_0 minimization using the Locally Competitive Algorithm architecture on the TrueNorth chip using our novel design methodology.

CHAPTER 6

SUMMARY, DISCUSSION, AND FUTURE DIRECTIONS

6.1 Summary

The main goal of this work is to establish a meaningful framework for implementing high-precision, recurrent network architectures on the low-precision, digital, spiking IBM Neurosynaptic System and other hardware platforms that have similar architectural constraints. We develop a novel design methodology to map these network architectures to the TrueNorth chip and use this methodology to implement a biologically-plausible sparse approximation solver as well as other sparsity-based probabilistic inference solvers on neuromorphic hardware.

Our novel design methodology maps the Locally Competitive Algorithm to the TrueNorth chip to solve for the sparse approximation of a signal, offering the largest LCA dictionaries implemented on neuromorphic hardware to date with perfect precision. We also explain methods to map other sparsity-based probabilistic inference problems onto the hardware. We describe the optimal way to achieve these high precision calculations by encoding and decoding the signal using large time windows. We discuss in detail functional processing units for use on the hardware that offer non-linear thresholds, increased vector-matrix multiplication precision, and the ability to accurately implement a recurrent network on the hardware. Our design methodology can be extended for use in embedded systems signal processing applications that may require similar functionality.

6.2 Discussion

6.2.1 Convergence Time

The time required to converge to the correct solution is not ideal for applications requiring real-time or near real-time calculations. One might speculate that future hardware iterations or hardware tailored to this implementation of the LCA will offer clock cycles faster than the one millisecond the TrueNorth chip offers. Therefore, we focus on the fact that regardless of how long our iterations might take, that our values for each iteration match the discrete LCA exactly.

6.2.2 Thermometer Encoding

Although we focus on accuracy over time-efficiency, we investigate thermometer encoding in efforts to speed up the system on the TrueNorth chip. To perform this encoding, an array of neurons with incremental thresholds is used for each evolving LCA variable. To compute u_m, each TrueNorth neuron in the array of neurons contains the full state of LCA node m as the state charges and suppresses over time. This is achieved by using a positive threshold of $\alpha = 0$ with a linear reset of $V_m = V_m - \alpha = V_m$ for each TrueNorth tick.

Upon spiking, the node state is therefore retained and the TrueNorth neurons start spiking in increasing order of thresholds as the node state grows larger. If a TrueNorth neuron within the array of neurons is saturating, the LCA node state has reached a value greater than that neuron's incremental threshold. If a TrueNorth neuron within this array is not saturated but has a constant spike rate, the LCA node state is between the threshold of that neuron and the previous neuron's threshold. While this method offers faster computation we find that the convergence of the LCA system is inaccurate and unreliable for use in our framework.

6.3 Future Directions

6.3.1 Alternate Dictionaries

Dictionaries with different properties than those used in our Locally Competitive Algorithm (LCA) system on the TrueNorth chip are useful for other signal processing applications. In some sparse coding literature, the dictionary consists of elements that capture common structures and patterns in the data [12], much like the localized, oriented, and bandpass receptive fields of the mammalian primary visual cortex [11, 23]. These fields can be accurately modeled by Gabor-like transforms [30]. For a dictionary to produce sparse representations that overcome the issues of changes in position, size, and orientation, an overcomplete set of vectors that consist of various dilations and translations of such receptive fields are effective [31].

These dictionaries take on a wider variety of values that may not be $\{$-1,0,1$\}$ as we use. The initial projection multiplication matrix values will take on larger values than 1 and therefore the values of the inhibition matrix would also grow larger and might require higher precision than 9-bits signed. This will require an increase in neurons per LCA node in each layer and therefore the number of cores required for the computation.

6.3.2 Other Problems of Interest

Using the LCA architecture via our processing units for training a Support Vector Machine (SVM) [32] offers a promising opportunity. This would enable a fundamental machine learning algorithm to be implemented on the TrueNorth chip. With future advances in hardware, this would be the framework for efficient, low-power, on-chip machine learning.

REFERENCES

[1] C. D. James, J. B. Aimone, N. E. Miner, C. M. Vineyard, F. H. Rothganger, K. D. Carlson, S. A. Mulder, T. J. Draelos, A. Faust, M. J. Marinella, *et al.*, "A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications," *Biologically Inspired Cognitive Architectures*, 2017.

[2] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, and Y. Nakamura, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[3] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," *Current opinion in neurobiology*, vol. 14, no. 4, pp. 481–487, 2004.

[4] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural computation*, vol. 20, no. 10, pp. 2526–2563, 2008.

[5] E. J. Cands, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *Information Theory, IEEE Transactions on*, vol. 52, no. 2, pp. 489–509, 2006.

[6] M. Zibulevsky and M. Elad, "L1-l2 optimization in signal and image processing," *Signal Processing Magazine, IEEE*, vol. 27, no. 3, pp. 76–88, 2010.

[7] A. Y. Yang, Z. Zhou, A. G. Balasubramanian, S. S. Sastry, and Y. Ma, "Fast-minimization algorithms for robust face recognition," *Image Processing, IEEE Transactions on*, vol. 22, no. 8, pp. 3234–3246, 2013.

[8] M. Elad, M. A. Figueiredo, and Y. Ma, "On the role of sparse and redundant representations in image processing," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 972–982, 2010.

[9] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[10] J. Sawada, F. Akopyan, A. S. Cassidy, B. Taba, M. V. Debole, P. Datta, R. Alvarez-Icaza, A. Amir, J. V. Arthur, and A. Andreopoulos, "Truenorth ecosystem for brain-inspired computing: Scalable systems, software, and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Press, p. 12, ISBN: 1467388157.

[11] D. J. Field, "What is the goal of sensory coding?" *Neural computation*, vol. 6, no. 4, pp. 559–601, 1994.

[12] B. A. Olshausen, "Principles of image representation in visual cortex," *The visual neurosciences*, vol. 2, pp. 1603–1615, 2003.

[13] A. Balavoine, C. J. Rozell, and J. Romberg, "Global convergence of the locally competitive algorithm," in *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011 IEEE*, IEEE, pp. 431–436, ISBN: 1612842275.

[14] A. Balavoine, J. Romberg, and C. J. Rozell, "Convergence and rate analysis of neural networks for sparse approximation," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 9, pp. 1377–1389, 2012.

[15] A. S. Cassidy, R. Alvarez-Icaza, F. Akopyan, J. Sawada, J. V. Arthur, P. A. Merolla, P. Datta, M. G. Tallada, B. Taba, and A. Andreopoulos, "Real-time scalable cortical computing at 46 giga-synaptic ops/watt with," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, IEEE Press, pp. 27–38, ISBN: 1479955000.

[16] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, and V. Feldman, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, IEEE, pp. 1–10, ISBN: 1467361283.

[17] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, and R. Alvarez-Icaza, "Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, IEEE, pp. 1–10, ISBN: 1467361283.

[18] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, IEEE Computer Society Press, p. 54, ISBN: 1467308048.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, ACM, pp. 675–678, ISBN: 1450330630.

[20] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, pp. 1117–1125.

[21] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, and D. R. Barch, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, p. 201 604 850, 2016.

[22] A. Vedaldi and K. Lenc, "Matconvnet: Convolutional neural networks for matlab," in *Proceedings of the 23rd ACM international conference on Multimedia*, ACM, pp. 689–692, ISBN: 1450334598.

[23] B. A. Olshausen, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[24] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[25] D. L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via 1 minimization," *Proceedings of the National Academy of Sciences*, vol. 100, no. 5, pp. 2197–2202, 2003.

[26] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM journal on scientific computing*, vol. 20, no. 1, pp. 33–61, 1998.

[27] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[28] A. S. Charles, P. Garrigues, and C. J. Rozell, "A common network architecture efficiently implements a variety of sparsity-based inference problems," *Neural computation*, vol. 24, no. 12, pp. 3317–3339, 2012.

[29] E. J. Candes, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted 1 minimization," *Journal of Fourier analysis and applications*, vol. 14, no. 5-6, pp. 877–905, 2008.

[30] D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells," *JOSA A*, vol. 4, no. 12, pp. 2379–2394, 1987.

[31] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, "Shiftable multiscale transforms," *Information Theory, IEEE Transactions on*, vol. 38, no. 2, pp. 587–607, 1992.

[32] F. Girosi, "An equivalence between sparse approximation and support vector machines," *Neural computation*, vol. 10, no. 6, pp. 1455–1480, 1998.